VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Konfigurační modul pro firmu Kvados, a.s.

Configuration Modul for Company Kvados, a.s.

2010 Pavel Vyletělek

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 31. března 2010 .....

Rád bych na tomto místě poděkoval vedoucímu práce Ing. Marku Běhálkovi za odborné vedení diplomové práce. Dále bych chtěl poděkovat Ing. Martinu Bajgarovi za obětovaný čas a vstřícnost a Martině Smetkové za pomoc při zpracování textu.

Abstrakt

Hlavním obsahem práce je vylepšit existující systém konfigurací klientů firmy KVADOS. Základem konfigurací jsou parametry, které mají místo hodnot výrazy. Tyto výrazy dnes překládá překladač s využitím regulárních výrazů. V této práci jsou místo regulárních výrazů použity bezkontextové gramatiky s formátem zápisu BNF. Navíc je dle požadavků do výrazů přidána možnost volat funkce. Pro testování vytváření konfigurací je vybudován systém složený ze serveru, klienta a webové služby. Server vytváří a modifikuje konfigurace. Webová služba má funkci prostředníka pro distribuci těchto konfigurací klientům. A klienti používají zmíněné konfigurace pro své správné fungování.

Klíčová slova: konfigurace, výraz, bezkontextová gramatika, BNF, webová služba, JSON

Abstract

The main purpose of this thesis is to improve the existing KVADOS company client configuration system. Recent solution uses parameters identified by expressions instead of values and a compiler that translates these expressions using regular expressions. In this thesis the context-free grammar in BNF format was used instead of the regular expressions. Furthermore, according to requirements, the ability for function calls was added into the expressions. To test the process of configuration creation a new system composing of server, client and Web service was built. Server creates and modifies the configurations. Web service distributes these configurations to clients. Clients adjust their behavior to the obtained configuration.

Keywords: Configuration, Expression, Context-free Grammar, BNF, Web Service, JSON

Seznam použitých zkratek a symbolů

ANTLR	–	ANother Tool for Language Recognition
BNF	–	Backus-Naur Form
CAB	–	CABinet
CLR	–	Common Language Runtime
EBNF	–	Extended BNF
GUI	–	Graphical User Interface
GUID	–	Globally Unique IDentifier
HTTP	–	HyperText Transfer Protocol
IDE	–	Integrated Development Environment
ISO	–	International Organization for Standardization
JSON	–	JavaScript Object Notation
OS	–	Operating System
PDA	–	Personal Digital Assistant
RPC	–	Remote Procedure Call
SMTP	–	Simple Mail Transfer Protocol
SOAP	–	Simple Object Access Protocol
SQL	–	Structured Query Language
UDDI	–	Universal Description, Discovery and Integration
UML	–	Unified Modeling Language
WCF	–	Windows Communication Foundation
WS	–	Web Service
WSDL	–	Web Services Description Language
XML	–	eXtensible Markup Language

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Jedno z mnoha řešení firmy Kvados je klient pro mobilní zařízení - typicky nějaké PDA zařízení. Tento klient je určen pro zjednodušení práce obchodníků v terénu, servisních techniků nebo marketingových tazatelů. Každý zákazník, jenž využívá tuto klientskou aplikaci, má odlišné požadavky. To vede k problému složité modifikovatelnosti klientů. Modifikovatelnost je v dnešní době řešena konfiguracemi nastavující klienty přesně dle požadavků zákazníků. Stejně jako klienti samotní, jsou i konfigurace vytvářeny na severu, který je rovněž spravuje. Jádrem problému je nedostačující úroveň konfigurovatelnosti těchto klientů. Tuto úroveň je nutné vhodným způsobem vylepšit, čímž se zabývá tato diplomová práce. A cílem práce je tedy vytvořit konfigurační modul, který má vést k lepšímu řešení vytváření konfigurací klientů.

V kapitole 2 jsou popsány všechny použité technologie a přístupy. Hlavně jsou popsány tři technologie: *webové služby, gramatiky a překladače* a *JSON*. U každé technologie je ve stručné podobě její představení a seznámení s principem fungování. V závěru každé technologie je vždy zmíněn způsob využití či začlenění do této diplomové práce. V kapitole 3 se nachází nástin aktuálního stavu, z čeho je systém složený a jak vlastně funguje. Další kapitola (kapitola 4) obsahuje zadání samotné práce. Poté jsou požadavky (kapitola 5) a cíle očekávané od výsledného systému a požadavky kladené na jednotlivé části. Valná většina těchto požadavků je specifikována zástupcem firmy Kvados. Na začátku jsou sepsány požadavky, poté jsou případy užití. Pak následuje kapitola 6 - analýza. V této části je celý problém podrobně rozebrán a obecně zpracováno řešení systému. Tato kapitola obsahuje dostatečné množství diagramů i obrázků pro lepší pochopení struktury a fungování vytvořeného softwarového systému. Předposlední kapitola obsahuje návrh a implementace. Zde se nachází návrh grafického uživatelského rozhraní, diagram nasazení, použité nástroje a provoz systému. V závěrečné kapitole zhodnocuji stav řešení a vlastní přínos práce. Na konci kapitoly je shrnutí možných vylepšení, zdokonalení nebo jiných možných řešení systému navržené zástupcem firmy, vedoucím práce či mnou samotným.

2 Použité technologie

V této kapitole jsou popsány všechny základní použité technologie. Hlavně jsou popsány tři technologie: *webové služby, gramatiky a překladače* a *JSON*.

2.1 Webové služby

Tato sekce obsahuje základní popis webových služeb. Kromě popisu se zde nachází také způsob začlenění v této diplomové práci.

2.1.1 Co je webová služba?

Webová služba (WS) je systém pro komunikaci mezi dvěma uzly (klient vs. webová služba) v internetové síti. Webová služba je popsána speciálním jazykem nazývaným WSDL. V popisu WS je rovněž popsán způsob komunikace s onou službou. Pakliže klient WS ví jak komunikovat, může zahájit komunikaci pomocí protokolu SOAP. Protokol SOAP specifikuje formát zpráv[1].

2.1.2 Princip fungování

Základní princip fungování webové služby je zobrazen na obrázku 1. Na daném obrázku se nachází tři základní bloky, které mezi sebou komunikují a vyměňují si příslušné zprávy.

Funkce jednotlivých bloků:

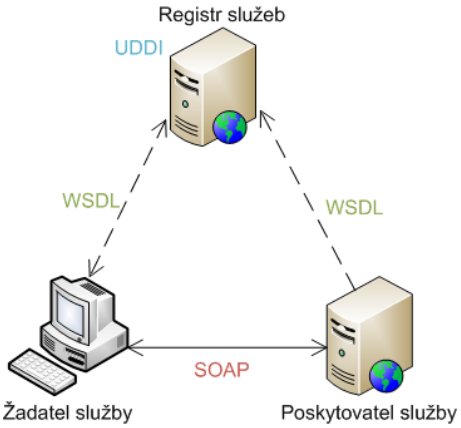
Registr služeb (Service Broker) - místo, kde jsou uloženy informace o webových službách a jejich poskytovatelích. Klienti zde vyhledávají poskytovatele. Registr služeb poskytuje informace potřebné pro navázání komunikace mezi uživatelem a poskytovatelem služeb.

Poskytovatel služby (Service Provider) - subjekt poskytující určitou webovou službu. Je to softwarová či hardwarová platforma zajišťující provoz všech webových služeb, které jsou u něj uloženy.

Žadatel služby (Service Requester) - klient mající zájem o využívání určité webové služby. Klient si nejprve vyhledá požadovanou službu v registru služeb. Z registru si získá popis webové služby a poté může službu začít používat. V podstatě se jedná o aplikaci, jež nemá vlastní rozhraní, které nahrazuje právě prostřednictvím webových služeb.

Základní pojmy:

SOAP - protokol pro výměnu zpráv založených na XML přes internetovou síť. Hlavně se používá v kombinaci s protokolem HTTP. SOAP tvoří základní vrstvu komunikace mezi registrem služeb, poskytovatelem služeb a klienty. Existuje několik šablon



Obrázek 1: WS schéma

pro komunikaci na protokolu SOAP. Nejznámější šablonou je RPC šablona, kdy se jedná o komunikaci mezi klientem a serverem. Server ihned odpovídá na požadavky klienta. Formát SOAP zprávy je na obrázku 2.

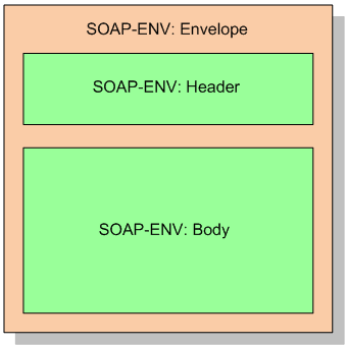
WSDL - popis webové služby včetně všech operací, které vystavuje a je možné je využívat. Formát zápisu je ve standardu XML. Slouží k popisu komunikace klienta s danou webovou službou. Podporované operace a zprávy jsou popsány abstraktně a potom se omezují na konkrétní síťový protokol a formát zprávy. WSDL je často používáno v kombinaci se SOAP a XML, pro poskytování webové služby po internetu. Klientská aplikace si přečte WSDL, aby si zjistila, jaké operace jsou dostupné u dané webové služby. Speciální datové typy jsou také uloženy ve WSDL souboru, který popisuje tuto WS.

UDDI - databáze poskytující mechanismy pro registrování a vyhledávání webových služeb. UDDI funguje jako adresář, který obsahuje informace o subjektech a službách poskytovaných těmito subjekty. Samotný registr pracuje rovněž jako webová služba a komunikace s ní tedy opět probíhá pomocí SOAP, potažmo WSDL[1].

Další informace týkající se webových služeb v .NET frameworku se nacházejí zde [2].

2.1.3 Alternativy

Ve většině případů se dá určitá technologie nahradit jinou technologií (popř. přístupem). Ani webové služby nejsou výjimkou. Jedna z alternativ je využití vzdáleného volání procedur. Vzdálené volání procedur neboli RPC je metoda starší než webové služby. Webové služby z něj v podstatě vycházejí. Jedná se o mechanismus volání nějaké funkce nebo metody, která je umístěna na jiném systému. Další alternativou WS je .NET WCF. Je to komunikační framework jako součást .NET frameworku. Jeho účelem je jednoduše vytvářet aplikace zaměřené na komunikaci přes web, v podnicích či korporacích. Podobných technologií však existuje celá řada.



Obrázek 2: SOAP schéma

2.1.4 Využití

Zvolený přístup má být použit k následujícímu účelu. Server vytváří konfigurace pro klienty. Konfigurace je nutné následně redistribuovat příslušným klientům. Pro tento účel se nejlépe hodí webové služby ze dvou hledisek. Za prvé je jedno, v jakém jazyce jsou klienti webové služby napsáni, což je velice důležité. Firmou KVADOS vytváření klienti jsou naprogramováni v C++ nebo v C#. Druhým důvodem je, že firma KVADOS již k jistým účelům webové služby používá. Je jednodušší rozšířit řešení stávající nežli vytvářet úplně nové řešení.

2.2 Překladače a gramatiky

Kapitola překladače a gramatiky zobrazuje obecný náhled na teorii gramatik a na nich založených překladačů. Také bude zmíněno alternativní řešení. A podobně jako u webových služeb začlenění gramatiky v diplomové práci.

2.2.1 Struktura překladače

Překladače mají dvě dílčí části. První část provádí analýzu zdrojového kódu (programu) a druhá část vytváří cílový kód (program). Analýza provádí rozklad zdrojového kódu (programu) na menší součásti. Tyto součásti jsou podrobeny různým kontrolám. Po úspěšné analýze se ze vzniklých součástí pomocí syntézy vytvoří cílový kód (program). Při provádění analýzy a syntézy se používá společná tabulka symbolů.

Jednotlivé kroky analýzy:

Lexikální analýza - zdrojový kód (program) je řetězec znaků, který je vstupem této analýzy. Vstupní řetězec je čten sekvenčně zleva doprava a během čtení je vytvářena tabulka symbolů (tokenů) - konstanty, identifikátory, klíčová slova, nebo operátory.

Syntaktická analýza - pomocí vytvořené tabulky symbolů se vytvoří hierarchicky zanořené struktury příslušného významu, např. výrazy, příkazy, deklarace či program.

Během syntaktické analýzy se provádí kontrola, zda jsou symboly použity ve zdrojovém kódu správně utvořeny.

Sémantická analýza - v průběhu této analýzy se provádí kontroly, zajišťující správnost programu z hlediska vazeb, které nelze provádět v rámci syntaktické analýzy (např. kontrola deklarací, typová kontrola)[4].

2.2.2 Bezkontextové gramatiky a EBNF

Gramatika je soupis pravidel popisujících syntaxi jazyka. Pro popis cílového jazyka v generátoru překladačů se používají bezkontextové gramatiky. Bezkontextová gramatika má jedno počáteční pravidlo neboli neterminál. Každé pravidlo může být přepsáno na více než jeden neterminál (nekoncové pravidlo) či terminál (koncové pravidlo).

Nejznámějším a hojně používaným zápisem gramatiky je BNF (Backus-Naur Form). Základní formát pravidel $a : b$ popisuje, že pravidlo a bude nahrazeno pravidlem b. Pokud je b pravidlo, pak bude nahrazeno pravou stranou. Počet a pořadí pravidel na pravé straně je nekonečný a libovolný. Více pravidel na pravé straně se odděluje symbolem |. Řetězce se uzavírají do jednoduchých uvozovek. Jelikož je v BNF těžké vytvořit opakování pravidel, používá se mnohem častěji upravený formát EBNF (Extended BNF). EBNF umožňuje vytvářet volitelná pravidla, opakování pravidel a závorky[7, 4].

Symboly	Popis
()	závorky se používají k seskupení pravidel, takže se tváří jako jeden celek
?	jakýkoliv jeden token za ? se opakuje 0 až 1 krát
$*$	jakýkoliv jeden token za $*$ se opakuje 0 až nekonečně krát
$+$	jakýkoliv jeden token za $+$ se opakuje 1 až nekonečně krát
.	jakýkoliv jeden znak/token
~	jakýkoliv jeden znak/token za ~ nemůže být na tomto místě
..	vkládá se mezi dva znaky a znamená rozsah znaků od-do včetně počátečního a koncového znaku

Tabulka 1: Symboly používané v EBNF

2.2.3 Alternativy

Vhodnou náhradou za bezkontextové gramatiky jsou regulární výrazy. Ačkoliv jsou regulární výrazy v chomského klasifikaci na úrovni nižší, přesto je možno využít regulární výrazy k vytvoření překladače. Ovšem pro překladače schopné rozpoznávat složitější výrazy a jazykové konstrukce se regulární výrazy na rozdíl od bezkontextových gramatik hodí méně, ne-li vůbec.

2.2.4 Využití

Parametry nacházející se v konfiguracích obsahují hodnotu vyjádřenou výrazem (bližší popis výrazů i samotných konfigurací je v kapitole 5). Aby bylo možné výrazy přeložit, je nutné mít lexikální a syntaktický analyzátor (respektive překladač). V současnosti jsou pro parsování výrazů využívány regulární výrazy. Požadavkem je vytvořit sofistikovanější překladač, který bude možno snáze rozšiřovat o další konstrukce. Zde se tedy více hodí použít bezkontextové gramatiky. Gramatika se bude vytvářet ve formátu BNF ve vhodném generátoru překladačů.

2.3 JSON

Tato kapitola obsahuje popis datového formátu JSON včetně jeho použití v diplomové práci.

2.3.1 Obecný popis

JSON je akronym pro výraz JavaScript Object Notation. Je to formát pro výměnu dat založený na podmnožině programovacího jazyka JavaScript (standard ECMA-262 třetí vydání - prosinec 1999). JSON je textový na jazyce zcela nezávislý formát. Využívá konvence dobře známé z programovacích jazyků rodiny C (C, C++, C#, Java, JavaScript, Perl, Python a další). To dělá z JSON ideální formát pro výměnu dat. Používá se spíše na výměnu menších dat.

JSON využívá dvě základní struktury:

Kolekce párů *název:hodnota* - v různých jazycích je realizována různě - objekt, záznam (record), struktura (struct), slovník (dictionary), hash tabulka, klíčovaný seznam (keyed list) nebo asociativní pole.

Tříděný seznam hodnot - ve většině jazyků realizován jako pole, vektor, seznam (list) nebo posloupnost (sequence).

Jedná se o běžné datové struktury a většina moderních programovacích jazyků je určitým způsobem podporuje. Z tohoto pohledu je velmi výhodné na nich založit jazykově nezávislý formát výměny dat [5]. Na webu `http://www.json.org/` jsou struktury používané při vytváření JSON dokumentu pěkně popsány. Rovněž se zde nachází spousty odkazů na jiné zdroje včetně odkazů na různé knihovny.

2.3.2 Alternativy

Nejznámější alternativou JSON formátu je již zmíněný formát XML. Oba dva jsou formáty pro výměnu dat. Výhodou JSON je snadná čitelnost, zapisovatelnost a analyzovatelnost nejen strojem, ale i člověkem. Naopak výhoda XML je při přenosu objemnějších dat. XML může pro výběr uzlů využívat dotazovací jazyk XPath, což JSON nemůže. Pro popis struktury XML se používá XML Schema, zatímco JSON nic podobného nemá. XML je

rovněž oproti JSON mnohem známější a rozšířenější. Navíc samotný .NET framework má knihovny pro práci s XML. Podrobnější informace jako JSON, tak o XML se nacházejí v knize [6].

2.3.3 Využití

Konfigurace je třeba ukládat v nějakém vhodném formátu. V současnosti je tento problém řešen pomocí textových souborů či registrů. Firma KVADOS zde nezadala nějaké konkrétní požadavky, avšak mělo by se jednat o jiný formát odlišný od aktuálního. Jako vhodnější řešení se zde zvolil JSON kvůli rychlejšímu zpracování při přenosu malých dat. Pro práci s formátem JSON existuje spoustu open source knihoven a v práci bude tedy použita nějaká z nich.

3 Aktualní stav

Kapitola má za cíl seznámit čtenáře s existujícím řešením upravovaného systému. Nejprve je to z pohledu architektury a poté z pohledu fungování.

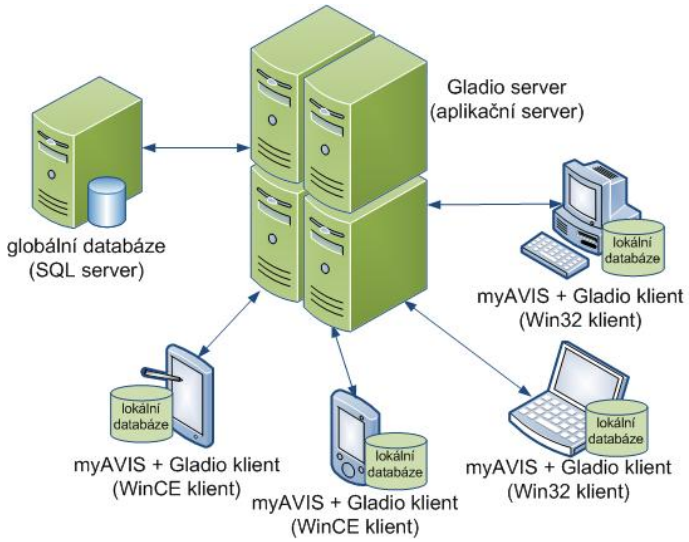
3.1 Architektura

V současné době je ve firmě KVADOS systém mající dvě základní části - server a klient. Klienti potažmo klientské aplikace jsou různé. Jde například o dříve zmíněný myAVIS, myCASH či další již existující nebo teprve vyvíjené aplikace. Každá klientská aplikace se zaměřuje na jinou oblast činnosti a má své specifické zaměření. Rovněž jejich implementace je úplně odlišná. Některé jsou implementovány pro Win32 platformu, jiné pro WinCE platformu a některé jsou dostupné pro obě platformy. Klienti pro platformu WinCE využívají pouze nativní běhové prostředí. Důvodem je vyšší výkon a rychlejší běh. Oproti tomu klienti běžící na platformě Win32 využívají buď nativní běhové prostředí nebo CLR.

Tato práce je zaměřena na klientskou aplikaci myAVIS. A následné úpravy se týkají pouze této aplikace. Ostatní klienti nejsou podstatní a nebudou v souvislosti s touto prací zmiňováni, avšak tato úprava bude v budoucnu rozšířena i pro ně. myAVIS je aplikace nativní a implementována je pro WinCE i Win32. Jeho zaměření je na obchodníky v terénu, servisní techniky či marketingové tazatele.

Výčet základních modulů myAVIS klienta:

- katalog zboží;
- stav skladu;
- kontakty;
- objednávky;
- záznamy o navštívených klientech;
- plánování tras;
- plánování úkolů;
- výkaz práce;
- možnost přidělení teritorií;
- správa servisních míst;
- správa technických zařízení.



Obrázek 3: Architektura stávajícího systému

Co se týče serveru, ten je pro všechny klienty společný. Tento server se jmenuje Gladio a jeho hlavním úkolem je synchronizace dat mezi klientskou a serverovou databází. Gladio se skládá ze dvou částí - Gladio klient a Gladio server. Jak Gladio klient, tak i Gladio server jsou vytvořeni v jazyce C# a tak je běhovým prostředím CLR. Gladio klient je utilita umístěná na klientovi. Gladio server běží na aplikačním serveru. Gladio klient se připojuje ke Gladio serveru a společně provádějí synchronizaci dat mezi klientskou (lokální) databází a serverovou (globální) databází. Gladio server má krom synchronizace na starost celou řadu dalších činností. Zjednodušenou architekturu systému zachycuje obrázek 3.

Výčet základních funkcí Gladio serveru:

- příprava nových verzí klienta myAVIS;
- vytváření instalací klienta myAVIS;
- vytváření a správa konfigurací všech klientů;
- ověřování klientů;
- synchronizace dat mezi klientskou a serverovou databází.

3.2 myAVIS a Gladio

Na serveru Gladio se spustí konfigurátor, ve kterém se přesně dle požadavků zákazníka vytvoří klient. Vyberou se moduly s odpovídajícím nastavením. Nakonfiguruje se grafické uživatelské rozhraní. Vytvoří se konfigurace klienta. Tyto konfigurace obsahují parametry pro různá nastavení. Místo konkrétních hodnot však obsahují výrazy, z nichž

se výsledné hodnoty získají jejich vyhodnocením. V současnosti se pro jejich vyhodnocení používají regulární výrazy. Pomocí regulárních výrazů jsou definovány následující základní konstrukce (operátory a operandy):

- hodnoty Boolean;
- hodnoty Integer;
- hodnoty Double;
- hodnoty String;
- hodnoty DateTime;
- logické operátory $(AND, OR, ==, !=, >, >=, <, <=)$;
- matematické operátory $(+, -, *, /, \%)$;
- proměnné.

Pakliže je v konfigurátoru vytvořen klient přesně podle požadavků zákazníka, tzv. generátor na serveru vygeneruje instalační verzi klienta myAVIS. Instalační verze obsahuje nejen klienta, ale rovněž další potřebné soubory včetně CAB souborů, utilit, knihoven a jiných.

Takto vytvořená instalační verze myAVIS se nakopíruje na cílový přístroj. Instalace na WinCE (PDA, SmartPhone, ...) se provádí automaticky po tvrdém resetu zařízení. Pokud se jedná o Win32 zařízení, reset se neprovádí, pouze se manuálně spustí instalace.

Po úspěšném dokončení instalace klientské aplikace myAVIS se uskuteční její první spuštění. Při prvním spuštění se vždy založí klientská (lokální) databáze. Poté se spustí Gladio klient a ten provede první (inicializační) synchronizaci s Gladio serverem. Když synchronizace skončí, je aplikaci myAVIS možné používat. Jelikož se mění data v lokální i globální databázi, je vhodné čas od času provést synchronizaci. Synchronizace nejsou automatické, musí se manuálně spouštět buď přímo z průzkumníka Windows anebo z aplikace myAVIS.

4 Specifikace zadání

Jedním z cílů práce je úprava výrazů obsažených v parametrech konfigurací. V podstatě by se měly využít stávající výrazy s množinou operátorů a operandů popsaných v předcházející kapitole. Jediným rozdílem bude jejich rozšíření o volání metod respektive funkcí. Syntaxe volání funkcí by měla být shodná se syntaxí volání funkcí v běžných objektově orientovaných programovacích jazycích (např. C#, C++, Java, ...). Funkce volané z výrazů budou uloženy v knihovně nazvané *runtime knihovna*.

Další změna oproti současnému řešení spočívá v nahrazení regulárních výrazů bezkontextovými gramatikami. Pro vytváření gramatik je zástupcem firmy požadováno využít speciální nástroj určený ke generování gramatik - tzv. generátor gramatik. Nástroj by měl být schopen vytvářet gramatiky ve standardizovaném formátu BNF, popř. EBNF. Je to proto, aby gramatika byla přehledná a mohla se snadno upravovat. Rovněž by měl nástroj umět vytvořené výrazy pro danou gramatiku otestovat. Z gramatiky se musí vygenerovat lexikální a syntaktický analyzátor v jazyce shodném s jazykem v jakém je vytvořeno Gladio (budou do něj v budoucnu začleněny). Tímto jazykem je C#. V současné době se používají pro překládání regulární výrazy. Ty překládají výrazy do objektové stromové struktury, kdy každý objekt reprezentuje buď jeden operand, nebo jeden operátor. Existující množinu tříd, ze kterých se objektová struktura vytváří, se musí vhodně upravit a rozšířit za účelem pokrytí nově definovaných výrazů. Pomocí obou analyzátorů se z výrazu vytvoří syntaktický strom a ten se převede do nové objektové stromové struktury. Na vytvořené stromové struktuře je poté možno provést vyhodnocení a získat tím požadovanou hodnotu určitého konfiguračního parametru.

Pro ověření fungování vytvářených gramatik se vytvoří funkční systém skládající se ze serveru a klienta. Úkolem serveru bude vytvářet či modifikovat konfigurace s využitím výrazů. Klient bude na jednoduchém uživatelském rozhraní prezentovat fungování konfigurací vytvořených serverem. K distribuci konfigurací ze serveru na klienta se doporučuje použít webových služeb, avšak není to nutností. Další nezbytné vlastnosti týkající se jednotlivých částí systému jsou shrnuty v bodech.

Zadání serveru pro práci s konfiguracemi:

1. implementace v jazyce C#;
2. jednoduché GUI;
3. přehledné zobrazování konfigurací uložených na serveru WS;
4. možnost modifikace existujících konfigurací uložených na serveru WS;
5. možnost vytváření nových konfigurací včetně jejich uložení na server WS;
6. součástí lexikální a syntaktický analyzátor;
7. alespoň jednoduchá kontrolu sémantiky;
8. součástí runtime knihovna s funkcemi, popř. i proměnnými využitými ve výrazech.

Zadání WS pro komunikaci a přístup ke konfiguracím:

1. implementace v jazyce C#;
2. vhodně zvolený způsob uložení konfigurací;
3. každá konfigurace v samotném souboru;
4. možnost načítání a ukládání konfigurací;
5. možnost stažení struktury (seznamu) úložiště konfigurací;
6. není nutné řešit odmazávání starých, již nepoužívaných konfigurací.

Zadání formátu konfigurací:

1. vhodně zvolený formát uložení (např. XML);
2. konfigurační parametry s výrazy místo hodnot;
3. operátory výrazu - logické, binární a matematické operátory;
4. operandy výrazu - celá čísla, desetinná čísla, řetězce, datum a čas, boolean;
5. proměnné operandy výrazu - proměnné a funkce z runtime knihovny;
6. rozlišení verzí;
7. identifikace konfigurací podle verze operačního systému, typu přístroje a zákazníka.

Zadání pro fungování klienta:

1. implementace v jazyce C++ nebo C#;
2. jednoduché GUI;
3. pravidelná kontrola aktualizace konfigurace s případným stažením nové verze;
4. prověření a zapracování nové konfigurace;
5. součástí runtime knihovna s funkcemi, popř. proměnnými využitými ve výrazech.

5 Specifikace požadavků

V této kapitole je soupis hlavních očekávaných vlastností systému, stejně jako popis jeho základních funkcí. Kapitola zahrnuje případy užití serveru a webové služby. Jako názorná ukázka jsou použity dva diagramy případů užití - 4 a 5. Jednotlivé případy užití zachycené na diagramech jsou ve stručné formě rozepsány.

5.1 Případy užití

Tato sekce obsahuje případy užití systému z pohledu serveru a z pohledu webové služby. Klient nemá, vzhledem ke své jednoduchosti, případy užití specifikovány.

5.1.1 Případy užití - server

Zobrazení struktury konfigurací:
Level: uživatelský cíl.
Klíčový uživatel: konzultant.
Stakeholders: programátor.
Předpoklady:

1. musí být dostupná webová služba;
2. musí existovat struktura konfigurací na serveru webové služby.

Minimální záruky: zobrazení chybové zprávy o nějaké neočekávané události.
Záruky úspěchu: přehledné zobrazení struktury konfigurací v serverovém GUI.
Popis:

1. server se připojí k webové službě;
2. server stáhne strukturu konfigurací nacházejících se na serveru webové služby;
3. server zobrazí strukturu konfigurací v GUI.

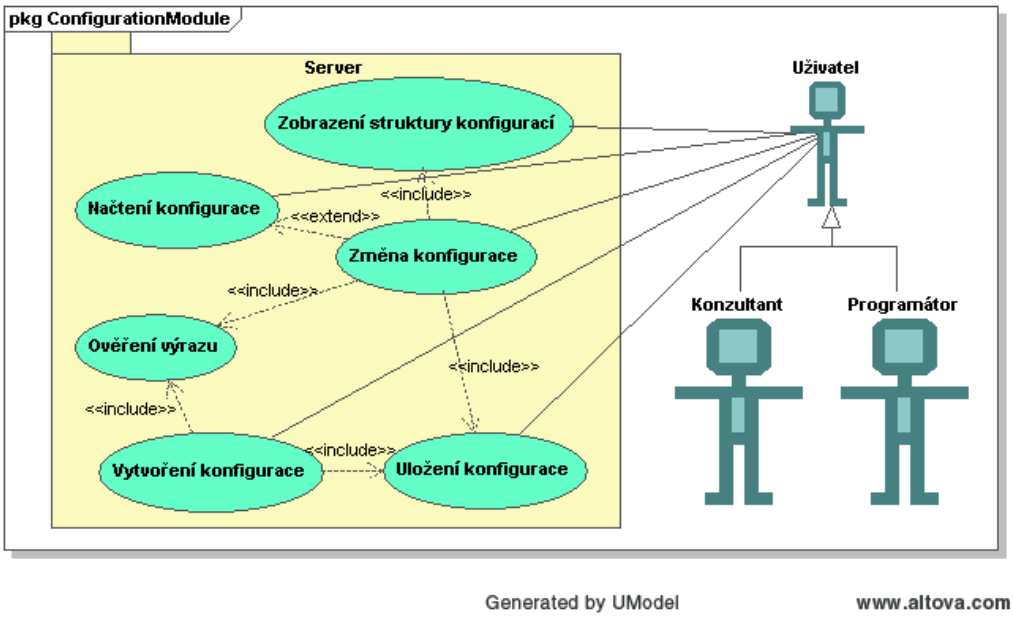
Rozšíření:

1a. nepodařilo se připojit k webové službě;

1a1. server zobrazí chybovou zprávu;

2a. chyba webové služby při stahování struktury konfigurací;

2a1. server zobrazí chybovou zprávu;
2a2. server ukončí stahování.



Obrázek 4: Případy užití - Server

Načtení konfigurace:
Level: uživatelský cíl.
Klíčový uživatel: konzultant.
Stakeholders: programátor.
Předpoklady:

1. musí být dostupná webová služba;
2. musí existovat alespoň jedna konfigurace na serveru webové služby.

Minimální záruky: zobrazení chybové zprávy o nějaké neočekávané události.
Záruky úspěchu: přehledné zobrazení hlavičky i parametrů načtené konfigurace v serverovém GUI.
Popis:

1. server se připojí k webové službě;
2. server načte existující konfiguraci ze serveru webové služby;
3. server zobrazí hlavičku i parametry načtené konfigurace v GUI.

Rozšíření:

1a. nepodařilo se připojit k webové službě;

1a1. server zobrazí chybovou zprávu;

2a. zvolená konfigurace na serveru webové služby neexistuje;

2a1. webová služba vrátí prázdnou konfiguraci;

2b. chyba webové služby při načítání konfigurace;

2b1. server zobrazí chybovou zprávu;
2b2. server ukončí stahování.

Změna konfigurace:
Level: uživatelský cíl.
Klíčový uživatel: konzultant.
Stakeholders: programátor.
Předpoklady:

1. musí být dostupná webová služba;
2. musí existovat struktura konfigurací na serveru webové služby;
3. musí existovat alespoň jedna konfigurace na serveru webové služby.

Minimální záruky: zobrazení chybové zprávy o nějaké neočekávané události.
Záruky úspěchu: úspěšná změna parametrů již existující konfigurace.
Popis:

1. server *stáhne strukturu konfigurací* nacházejících se na serveru webové služby;
2. konzultant vybere příslušnou konfiguraci;
3. server *načte existující konfiguraci* ze serveru webové služby;
4. konzultant modifikuje parametry konfigurace;
5. server *ověří výrazy* parametrů konfigurace;
6. server *uloží vzniklou konfiguraci* na server webové služby.

Ověření výrazu:
Level: vnitřní funkce.
Předpoklady:

1. server musí mít lexikální a syntaktický analyzátor;
2. server musí mít runtime knihovnu obsahující proměnné a funkce.

Minimální záruky: zobrazení chybové zprávy o nějaké neočekávané události.
Záruky úspěchu: úspěšné ověření správnosti všech zadaných výrazů.
Popis:

1. server provede lexikální analýzu;
2. server provede syntaktickou analýzu;
3. server zkontroluje existenci použitých proměnných a funkcí v knihovně;
4. server zkontroluje sémantiku.

Rozšíření:

1a. chyba během lexikální analýzy;

1a1. server zobrazí chybovou zprávu;

2a. chyba během syntaktické analýzy;

2a1. server zobrazí chybovou zprávu;

3a. proměnná nebo funkce nebyla nalezena;

3a1. server zobrazí chybovou zprávu;

4a. sémantická chyba;

4a1. server zobrazí chybovou zprávu.

Vytvoření konfigurace:
Level: uživatelský cíl.
Klíčový uživatel: konzultant.
Stakeholders: programátor.
Předpoklady: musí být dostupná webová služba.
Minimální záruky: zobrazení chybové zprávy o nějaké neočekávané události.
Záruky úspěchu: úspěšné vytvoření a uložení konfigurace na serveru webové služby.
Popis:

1. server vytvoří novou prázdnou konfiguraci;
2. konzultant vytvoří hlavičku konfigurace;
3. konzultant vloží všechny potřebné parametry do konfigurace;
4. server *ověří výrazy* parametrů konfigurace;
5. server *uloží vzniklou konfiguraci* na server webové služby.

Uložení konfigurace:
Level: uživatelský cíl.
Klíčový uživatel: konzultant.
Stakeholders: programátor.
Předpoklady: musí být dostupná webová služba.
Minimální záruky: zobrazení chybové zprávy o nějaké neočekávané události.
Záruky úspěchu: úspěšné uložení konfigurace na serveru webové služby.
Popis:

1. server se připojí k webové službě;
2. server uloží vytvořenou konfiguraci na serveru webové služby.

Rozšíření:

1a. nepodařilo se připojit k webové službě;

1a1. server zobrazí chybovou zprávu;

2a. zvolená konfigurace na serveru webové služby již existuje;

2a1. webová služba přepíše existující konfiguraci.

5.1.2 Případy užití - WS

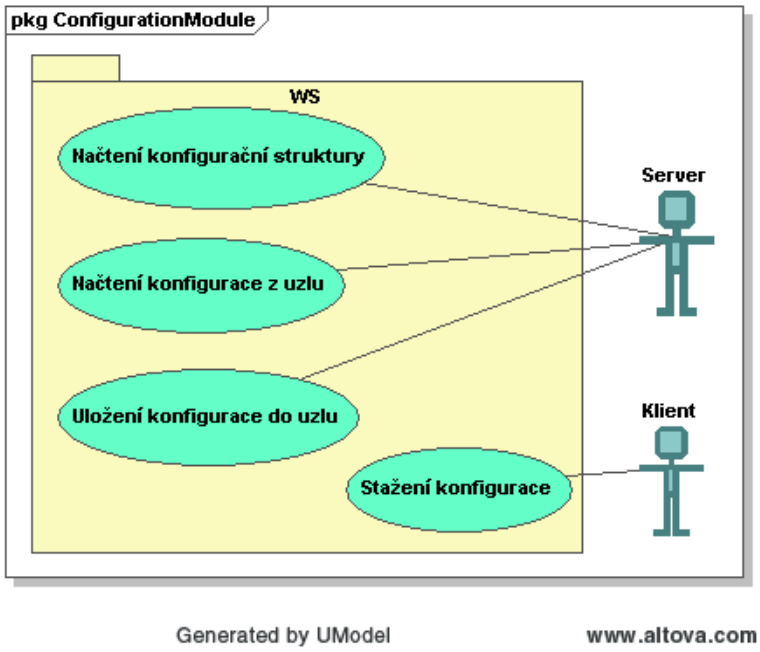
Načtení konfigurační struktury:
Level: uživatelský cíl.
Klíčový uživatel: server.
Minimální záruky: WS vrátí prázdnou zprávu.
Záruky úspěchu: úspěšné načtení struktury konfigurací.
Popis:

1. WS prochází úložiště konfigurací;
2. WS postupně vytváří strukturu konfigurací.

Rozšíření:

2a. nepodařilo se vytvořit strukturu konfigurací;

2a1. WS vrátí prázdnou strukturu.



Obrázek 5: Případy užití - WS

Načtení konfigurace z uzlu:
Level: uživatelský cíl.
Klíčový uživatel: server.
Předpoklady: musí požadovaná konfigurace existovat.
Minimální záruky: WS vrátí prázdnou zprávu.
Záruky úspěchu: úspěšné načtení požadované konfigurace.
Popis:

1. WS prochází úložiště konfigurací;
2. WS hledá konfiguraci podle specifikované hlavičky konfigurace;
3. WS načte danou konfiguraci.

Rozšíření:

2a. nepodařilo se najít konfiguraci;

2a1. WS vrátí prázdnou konfiguraci.

Uložení konfigurace do uzlu:
Level: uživatelský cíl.
Klíčový uživatel: server.

Minimální záruky: WS vrátí prázdnou zprávu.
Záruky úspěchu: úspěšné uložení dané konfigurace.
Popis:

1. WS prochází úložiště konfigurací;
2. WS hledá příslušné místo pro uložení dané konfigurace dle hlavičky z konfigurace;
3. WS uloží danou konfiguraci na příslušné místo.

Rozšíření:

2a. nepodařilo se najít příslušné místo pro uložení dané konfigurace;

2a1. WS neuloží danou konfiguraci;

3a. daná konfigurace již existuje;

3a1. WS přepíše existující konfiguraci.

Stažení konfigurace:
Level: uživatelský cíl.
Klíčový uživatel: klient.
Předpoklady: musí požadovaná konfigurace existovat.
Minimální záruky: WS vrátí prázdnou zprávu.
Záruky úspěchu: úspěšné stažení konfigurace klientem.
Popis:

1. WS prochází úložiště konfigurací;
2. WS hledá konfiguraci podle specifikované hlavičky konfigurace;
3. WS sestaví úplnou konfiguraci ze specifikované hlavičky a z nalezených parametrů konfigurace.

Rozšíření:

2a. nepodařilo se najít konfiguraci;

2a1. WS vrátí prázdnou konfiguraci;

3a. nepodařilo se sestavit konfiguraci;

3a1. WS vrátí prázdnou konfiguraci.

#	Název	Priorita	Technická obtížnost
1	Zobrazení struktury konfigurací	3	lehká
2	Načtení konfigurace	2	normální
3	Změna konfigurace	1	normální
4	Ověření výrazu	1	obtížná
5	Vytvoření konfigurace	1	normální
6	Uložení konfigurace	2	normální
7	Načtení konfigurační struktury	3	normální
8	Načtení konfigurace z uzlu	2	obtížná
9	Uložení konfigurace do uzlu	2	obtížná
10	Stažení konfigurace	1	obtížná

Tabulka 2: Seznam případů užití

Priority:

1 - vysoká priorita;

2 - střední priorita;

3 - nízká priorita.

Technické obtížnosti:

lehká - nejjednodušší případy užití;

normální - případy užití střední obtížnosti;

obtížná - nejkomplikovanější případy užití.

6 Analýza

Tato kapitola má za cíl prostudovat problematiku, provést analýzu a zpracování zadaného problému. Snahou je rozebrat specifikované požadavky a obecněji popsat řešení systému. Kapitola neobsahuje detailní a úplný popis analýzy, která by se v případě unifikovaného procesu vývoje aplikací prováděla. Je zaměřena pouze na vybrané části. Jedná se převážně o takové části, které jsou dosti komplikované, a je vhodné provést jejich analýzu. Nebo to mohou být partie nejpodstatnější z hlediska vyvíjeného systému. Pohled na budoucí systém je zjednodušeně zachycen na obrázku 6.

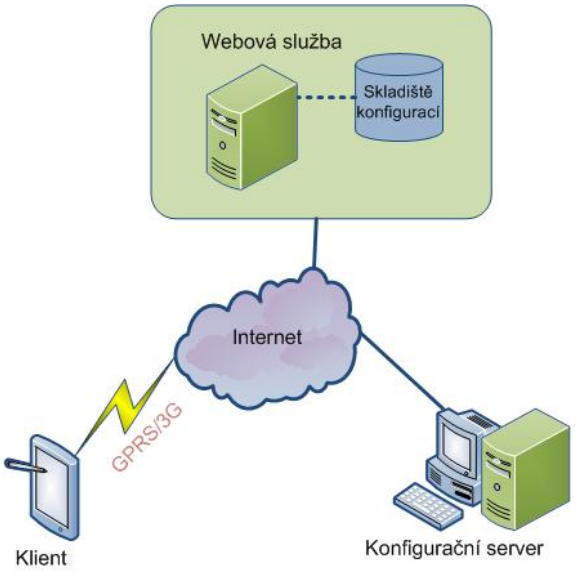
6.1 Statická analýza

Jak již bylo specifikováno v požadavcích, systém se bude skládat ze tří hlavních částí. Mimo těchto tří částí bude vytvořena jedna společná knihovna. Knihovna bude obsahovat třídy a funkce společné všem třem hlavním částem. Pro každou část včetně společné knihovny a konfigurací bude provedena statická analýza. Z důvodu omezeného rozsahu práce však bude statická analýza zaměřena pouze na určité části, které budou podrobněji rozebrány. Týká se to serveru, jenž se jakožto nejdůležitější prvek celého systému stará o vytváření a modifikaci konfigurací. Detailní rozbor bude proveden také u společné knihovny, která obsahuje velice podstatné třídy a funkce používané napříč systémem. Třídní diagram na obrázku 7 popisuje úplnou strukturu systému.

6.1.1 Statická analýza - server

Server bude v balíčku *ConfigurationServer*. Jeho úkolem bude vytváření nebo modifikace konfigurací pomocí grafického uživatelského rozhraní se dvěma obrazovkami. Hlavní třída *ServerStart* zobrazí po spuštění hlavní obrazovku (*MainScreen*). Na této obrazovce se bude zobrazovat strom konfigurací. Načtení stromu se provede vždy po startu serveru. Při každém přidání nové konfigurace se strom načte znova. Na hlavní obrazovce budou kromě stromu konfigurací také zobrazena hlavička a parametry konfigurace. V hlavičce bude možné upravovat jednotlivé parametry, ale nebude možné měnit verze. O verzování se bude starat webová služba. Parametry konfigurace se budou upravovat v jiném okně určeném přesně k tomuto účelu. Popis funkcí hlavní obrazovky:

1. Stažení stromu konfigurací z webové služby a její zobrazení v ovládacím prvku schopném zobrazit stromovou strukturu.
2. Stažení vybrané konfigurace z webové služby a zobrazení hlavičky i parametrů konfigurace v hlavním okně.
3. Smazání všech hodnot zadaných do hlavičky a všech parametrů aktuální konfigurace - vymazání okna.
4. Uložení vytvořené hlavičky a zadaných parametrů konfigurace na server webové služby.



Obrázek 6: Pohled na vyvíjený systém

5. Zobrazení okna *ExpressionScreen* při pokusu o změnu libovolného parametru konfigurace.

6. Při uzavření okna *ExpressionScreen* uložení vytvářeného či modifikovaného parametru do konfigurace.

Druhá obrazovka (*ExpressionScreen*) bude pro úpravu a vytváření jednotlivých parametrů konfigurace. V tomto okně se bude zadávat název parametru, jeho typ a nakonec hodnota. Konfigurace samotné budou popsány v jedné z následujících sekcí. Pro práci s konfiguracemi budou potřebné třídy uloženy ve společné knihovně. Detailní rozbor i s třídními diagramy bude rovněž v jedné z následujících sekcí. Základní funkce této obrazovky budou následující:

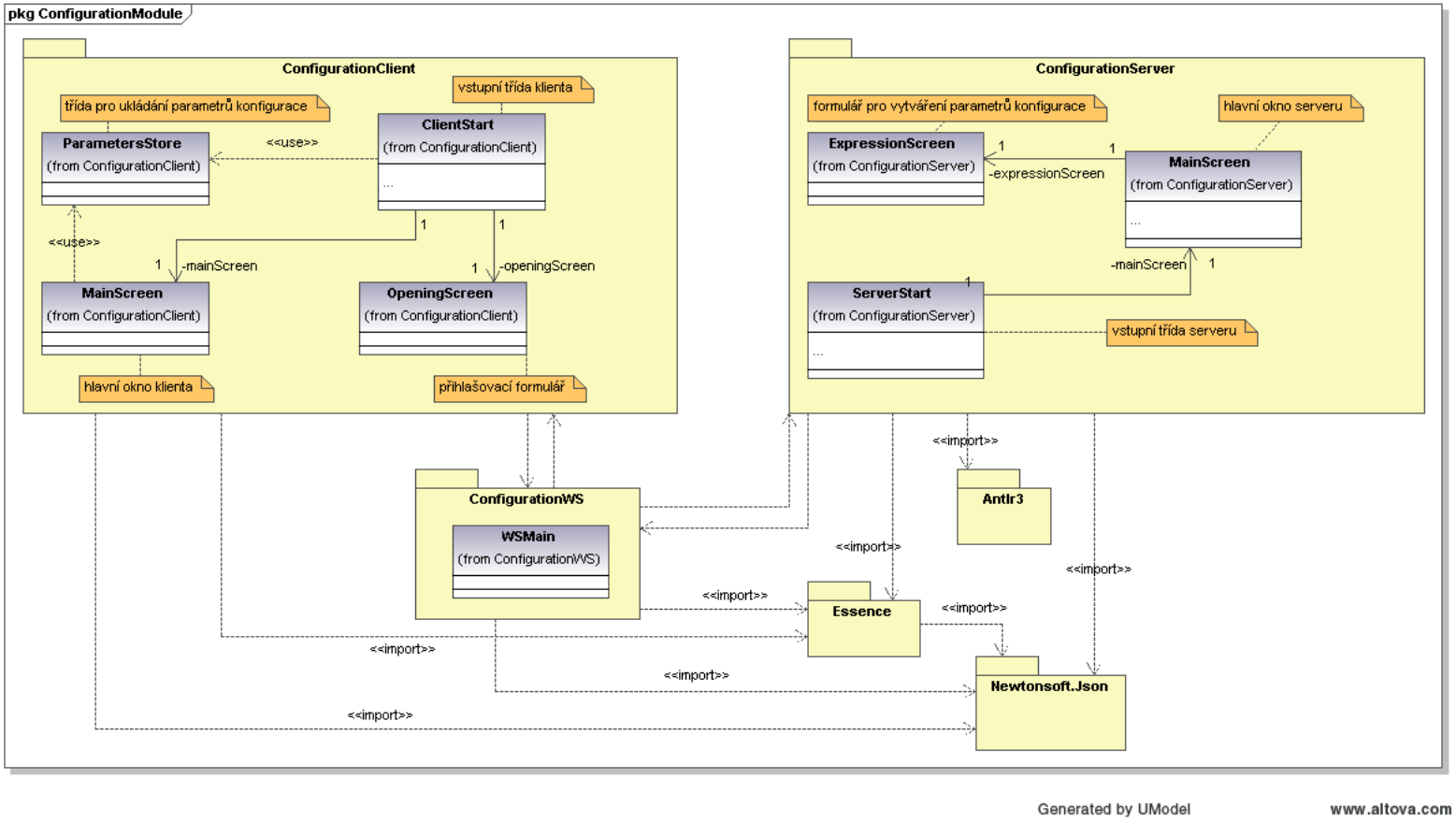
1. Převedení syntaktického stromu na stromovou strukturu objektů výrazu.

2. Vytvoření nového operandu z textové hodnoty.

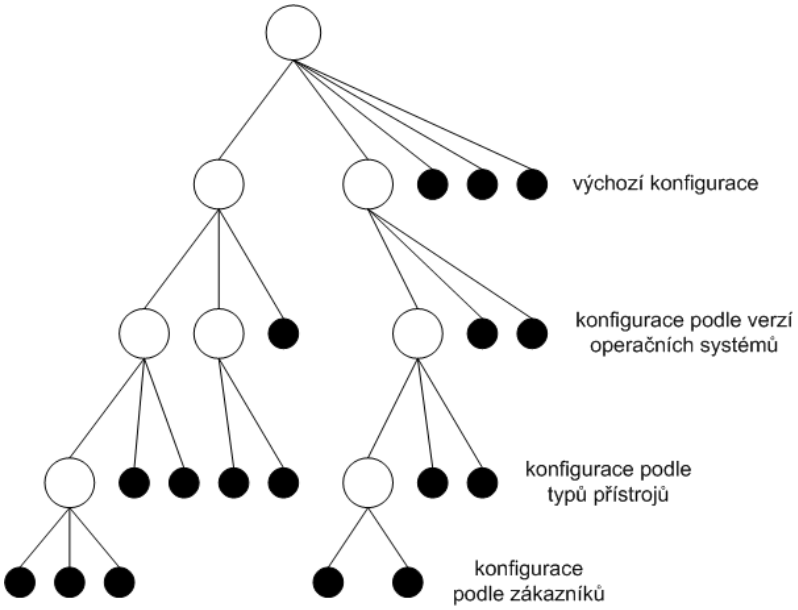
3. Provedení lexikální analýzy, syntaktické analýzy, vytvoření syntaktického stromu a sémantická kontrola.

4. Uložení vytvářeného či modifikovaného parametru do konfigurace.

Obě obrazovky serveru zde slouží jen pro testování konfigurací. V budoucnu nebude použit celý server, ale použijí se jen součásti pro práci s konfiguracemi a vytváření výrazů. A tyto součásti se zabudují do jádra existujícího Gladio serveru, kde nahradí v současnosti již nedostačující řešení.



Obrázek 7: Třídní diagram vyvíjeného systému



Obrázek 8: Stromová struktura konfigurací uložená na serveru webové služby

6.1.2 Statická analýza - WS

Webová služba (balíček *ConfigurationWS*) bude skladovat všechny konfigurace. Skladovat se budou do adresářové stromové struktury. Pro představu je její model na schématu 8. V adresářích budou uloženy soubory s konfiguracemi. Vždy jedna konfigurace v jednom souboru. Pojmenování souborů bude následující: $c\_[verze]$. Číslování verzí bude začínat na hodnotě jedna a bude se automaticky inkrementálně o jednu zvedat. V listech stromu budou soubory s číslem verze nula, kde bude uložena hlavička poslední nefunkční konfigurace pro daného klienta. Webová služba bude mít ve třídě *WSMain* tyto čtyři veřejné metody:

GetConfiguration - webová metoda pro stažení úplné konfigurace klientem.

GetNodeConfiguration - webová metoda pro stažení konfigurace z vybraného uzlu stromu konfigurací.

SetNodeConfiguration - webová metoda pro uložení konfigurace do vybraného uzlu stromu konfigurací.

GetConfigurationTree - webová metoda pro stažení struktury stromu konfigurací.

6.1.3 Statická analýza - klient

Klient bude v balíčku *ConfigurationClient*. Hlavní třída *ClientStart* bude po spuštění provádět stažení nejnovější konfigurace z webové služby, pokud to nebude možné, použije

svoji původní konfiguraci. Po stažení konfigurace se zobrazí obrazovka pro přihlášení (*OpeningScreen*). Po přihlášení se zobrazí okno aplikace (*MainScreen*). Pro ukládání aktuálních parametrů konfigurace načtených buď z původní konfigurace, anebo z konfigurace stažené z webové služby, se vždy použije třída *ParametersStore*. Klient nemá žádný zásadní význam pro budoucí použití. V systému bude pouze pro ukázku fungování konfigurací vytvářených serverem a jejich stahování z webové služby. Klient bude mít následující množinu parametrů v konfiguraci:

#	Parametr	Typ	Význam
1	OverallPrice	Double	vypočtená celková cena
2	OverallDiscount	Double	vypočtená celková sleva
3	CustomerDiscount	Integer	sleva pro daného zákazníka v procentech
4	ReportOverallTime	String	vypočtená délka pracovní doby
5	ReportBreakDuration	String	délka přestávky
6	ReportEndTime	String	konec pracovní doby
7	ReportStartTime	String	počátek pracovní doby
8	InvoiceTerm	DateTime	datum splatnosti faktury
9	PaymentMethod	String	způsob platby fakturace
10	WriteLog	Boolean	zda se bude vytvářet logovací výstup
11	ShowLogin	Boolean	zda se zobrazí přihlašovací obrazovka

Tabulka 3: Parametry použité v konfiguraci klienta

Podrobnější analýza klienta není zapotřebí, klient totiž nebude provádět žádnou činnost, pouze zpracuje a zobrazí konfiguraci. Místo tohoto klienta bude v reálném provozu používán existující klient.

6.1.4 Statická analýza - konfigurace

Formátem pro uložení byl po úvahách a konzultacích zvolen JSON. Detailnější informace použité technologie JSON se nacházejí v kapitole 2.3. Konfigurace se budou skládat ze dvou částí. Na začátku každé konfigurace bude hlavička (Header) obsahující identifikaci *verze operačního systému, typu přístroje a zákazníka*. U všech těchto atributů se bude uvádět verze, kvůli rozpoznání konfigurací, které byly použity ke složení konfigurace. V hlavičce bude také verze výchozí konfigurace, tedy taková, ze které celá sestavená konfigurace vychází.

Za hlavičkou bude následovat sekce parametrů (Parameters). Formát jednotlivých parametrů vkládaných do konfigurace bude tento: *název parametru (Name), textová hodnota (StringValue) a serializovaná hodnota (ExpressionValue)*. Název parametru bude mít dvě části oddělené podtržítkem. První část identifikace typu a druhá část vlastní název parametru. Textovou hodnotou bude původní výraz zadaný v textové podobě do konfigurace. Serializovanou hodnotou bude serializovaná stromová struktura objektů vytvořená přeložením textového výrazu pomocí překladače (lexikálního a syntaktického analyzátoru).

#	Atribut	Význam
1	DefaultVariant	verze výchozí konfigurace
2	OS	verze operačního systému
3	OSVariant	verze konfigurace pro daný operační systém
4	DeviceType	typ přístroje
5	DeviceVariant	verze konfigurace pro daný typ přístroje
6	CustomerId	identifikace zákazníka
7	CustomerVariant	verze konfigurace pro daného zákazníka

Tabulka 4: Atributy hlavičky

#	Identifikátor	Typ
1	b	Boolean
2	s	String
3	t	DateTime
4	n	Integer
5	d	Double

Tabulka 5: Seznam použitých typů v parametrech konfigurací

6.1.5 Statická analýza - knihovna Essence

Knihovna obsahující základní kód společný všem částem systému. Obsahuje třídy pro práci s konfiguracemi, serializací a s výrazy. Dále jsou všechny třídy a základní funkce zachyceny na okomentovaných třídních diagramech.

Strom konfigurací je třeba pro přenos ukládat do struktury objektů. Tato struktura je složena z objektů, které jsou instancí třídy *Node*.

Node
Popis: Třída pro uložení uzlu stromu konfigurací.
Vlastnosti:

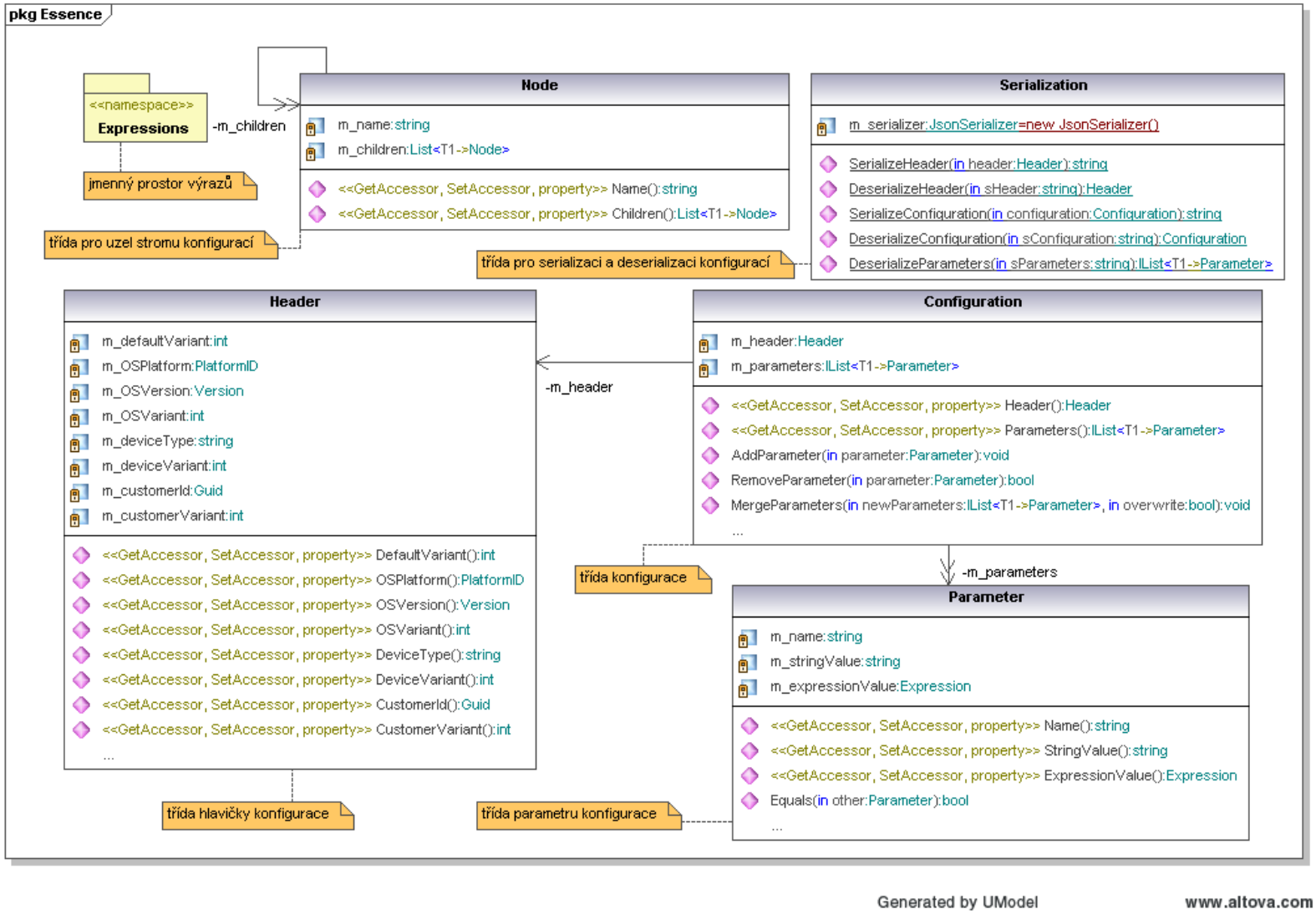
Name - jméno uzlu.

Children - pole potomků tohoto uzlu.

Pro serializaci a deserializaci konfigurací do JSON formátu slouží třída *Serialization*. Všechny její funkce pro serializaci či deserializaci hlaviček, parametrů, ale i celých konfigurací jsou napsány zde.

Serialization
Popis: Třída pro serializaci a deserializaci konfigurací.
Metody:

SerializeHeader - serializace hlavičky konfigurace.



Obrázek 9: Třídní diagram - knihovna Essence

DeserializeHeader - deserializace hlavičky konfigurace.

SerializeConfiguration - serializace celé konfigurace.

DeserializeConfiguration - deserializace celé konfigurace.

DeserializeParameters - deserializace parametrů konfigurace.

Třídy nacházející se na následných řádcích mají za úkol ukládat hlavičky konfigurací, parametry konfigurací i samotné konfigurace.

Header
Popis: Třída hlavičky konfigurace.
Vlastnosti:

DefaultVariant - verze výchozí konfigurace.

OSPlatform - typ operačního systému.

OSVersion - verze operačního systému.

OSVariant - verze konfigurace pro danou verzi operačního systému.

DeviceType - typ zařízení.

DeviceVariant - verze konfigurace pro daný typ zařízení.

CustomerId - identifikace zákazníka.

CustomerVariant - verze konfigurace pro daného zákazníka.

Parameter
Popis: Třída parametru konfigurace.
Vlastnosti:

Name - jméno parametru.

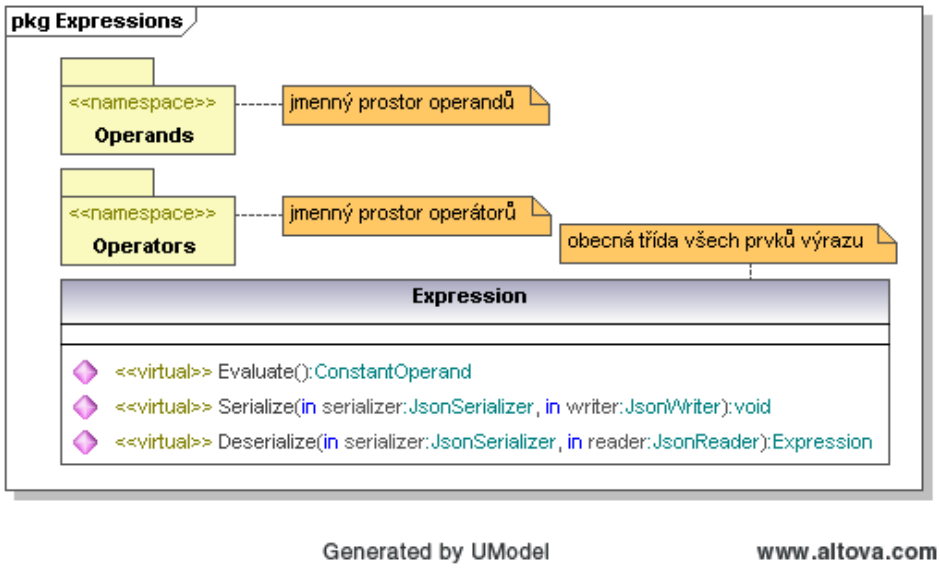
StringValue - výraz v textovém formátu.

ExpressionValue - výraz v objektové struktuře.

Metody:

Equals - porovnání, zda je jméno tohoto parametru shodné se jménem parametru zadaném v atributu metody.

Configuration
Popis: Třída konfigurace.
Vlastnosti:



Obrázek 10: Třídní diagram - výrazy

Header - hlavička konfigurace.

Parameters - kolekce parametrů konfigurace.

Metody:

AddParameter - přidání parametru do kolekce.

RemoveParameter - odebrání parametru z kolekce.

MergeParameters - spojení kolekce parametrů této instance s kolekcí parametrů zadanou v atributu metody. Při shodě jmen parametrů se ponechá původní hodnota nebo se zapíše hodnota nová v závislosti na druhém atributu.

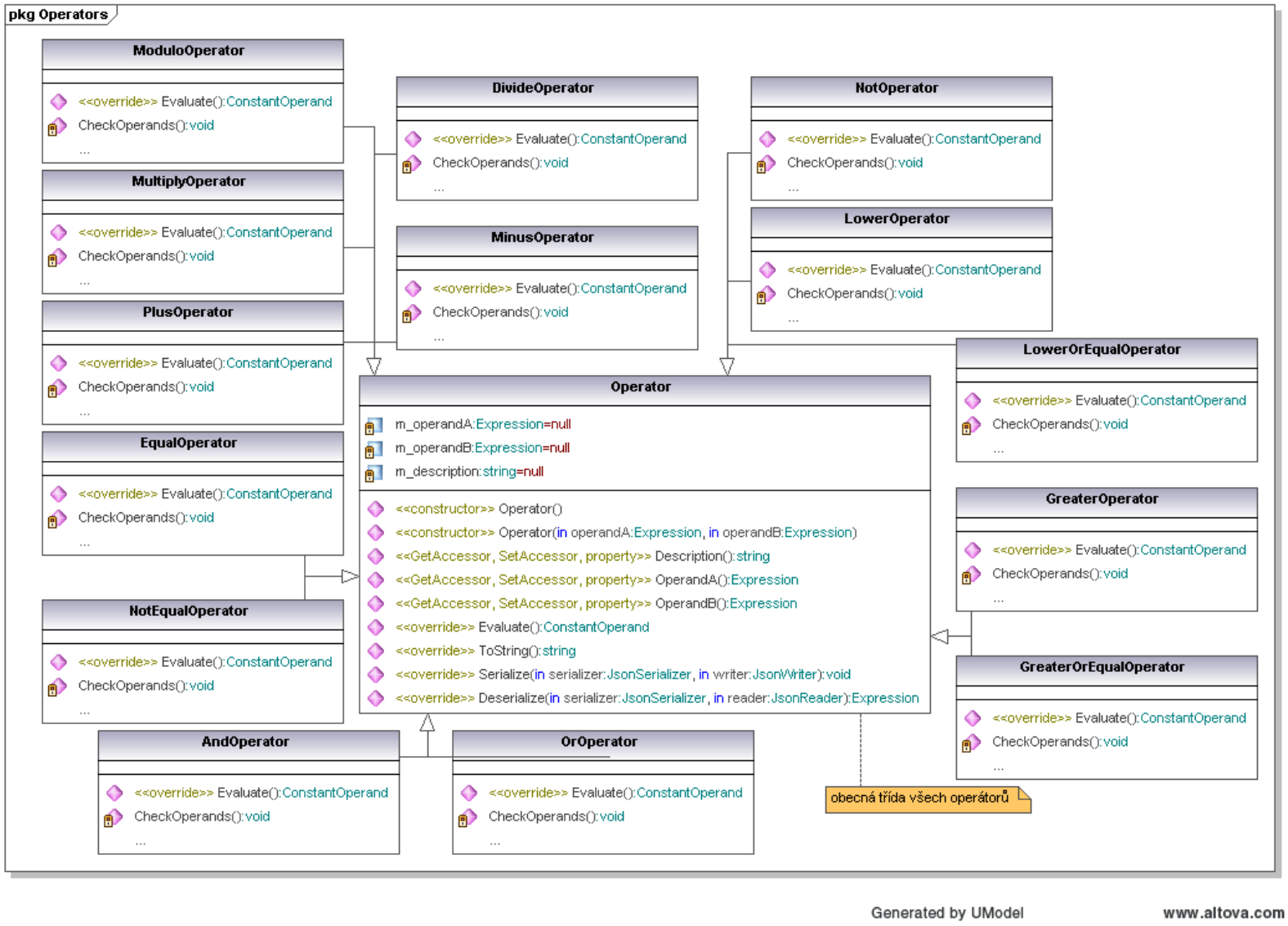
Výrazy se skládají z operátorů a operandů. Jejich obecný předek se nazývá *Expression* a má metody pro vyhodnocení, serializaci a deserializaci.

Expression
Popis: Výchozí třída pro všechny operátory a operandy.
Metody:

Evaluate - vyhodnocení výrazu.

Serialize - serializace výrazu.

Deserialize - deserializace výrazu.



Obrázek 11: Třídní diagram - operátory

Dále následují třídy nutné pro zpracování operátorů výrazu, jež mají jako výchozího předka *Operator* třídu.

Operator
Popis: Obecná třída všech operátorů.
Vlastnosti:

Description - popis.

OperandA - první operand.

OperandB - druhý operand.

Metody:

ToString - zobrazení operátoru v textové podobě.

Všechny operátory mají shodné metody, které se samozřejmě liší implementací podle významu operátoru. Metody jsou tyto:

Evaluate - vyhodnocení daného operátoru (výrazu).

CheckOperands - ověření typů obou operandů pro daný operátor.

Následuje výčet a popis všech operátorů, i když by účel měl být zřejmý již z názvu každého operátoru.

ModuloOperator
Popis: Operátor modulo.

DivideOperator
Popis: Operátor dělení.

MultiplyOperator
Popis: Operátor násobení.

MinusOperator
Popis: Operátor odečítání.

PlusOperator
Popis: Operátor sčítání.

NotOperator
Popis: Operátor negace.

LowerOperator
Popis: Operátor menší.

LowerOrEqualOperator
Popis: Operátor menší nebo rovno.

GreaterOperator
Popis: Operátor větší.

GreaterOrEqualOperator
Popis: Operátor větší nebo rovno.

EqualOperator
Popis: Operátor rovno.

NotEqualOperator
Popis: Operátor nerovná se.

AndOperator
Popis: Operátor konjunkce.

OrOperator
Popis: Operátor disjunkce.

Na následujících řádcích jsou shrnuty třídy nutné pro uchování hodnot operandů výrazu. Základní třída operandů je *Operand*.

Operand
Popis: Obecná třída všech operandů.
Vlastnosti:

Description - popis.

Metody:

CreateEmptyConstantOperand - vytvoření instance třídy ConstantOperand.

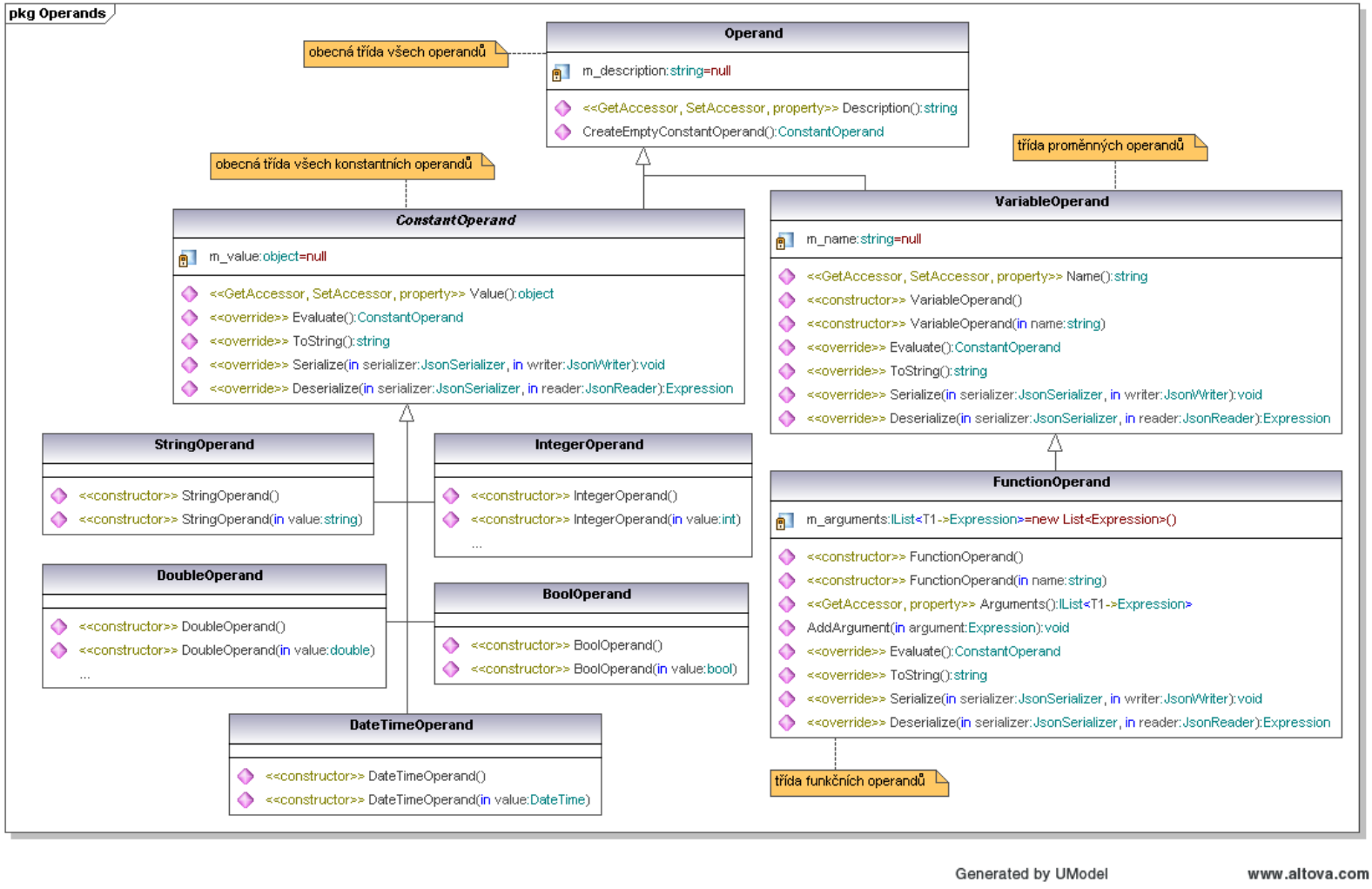
Operátory obsahující konstantní hodnotu mají společného předka - *ConstantOperand*. Za touto třídou následuje soupis všech typů operandů.

ConstantOperand
Popis: Obecná třída všech konstantních operandů.
Vlastnosti:

Value - hodnota operandu.

Metody:

ToString - zobrazení operandu v textové podobě.



Obrázek 12: Třídní diagram - operandy

StringOperand
Popis: Operand s textovou hodnotou.

IntegerOperand
Popis: Operand s celočíselnou hodnotou.

DoubleOperand
Popis: Operand s reálnou hodnotou.

BoolOperand
Popis: Operand s booleovskou hodnotou.

DateTimeOperand
Popis: Operand s datumovou hodnotou.

Další operand slouží pro uchování názvů proměnných, které mají svou hodnotu uloženou na klientovi. Klient si při vyhodnocení jeho hodnotu načte z dané proměnné.

VariableOperand
Popis: Třída proměnných operandů.
Vlastnosti:

Name - jméno proměnné.

Metody:

ToString - zobrazení operandu v textové podobě.

Poslední operand uchovává název a argumenty metody, z níž se získá hodnota jejím vyhodnocením. Klient při vyhodnocení zavolá metodu s daným názvem a předá mu specifikované atributy. Metody jsou uloženy v runtime knihovně, kterou má každý klient u sebe.

FunctionOperand
Popis: Třída funkčních operandů.
Vlastnosti:

Arguments - kolekce argumentů funkce.

Metody:

AddArgument - přidání argumentu do kolekce argumentů.

ToString - zobrazení operandu v textové podobě.

6.2 Dynamická analýza

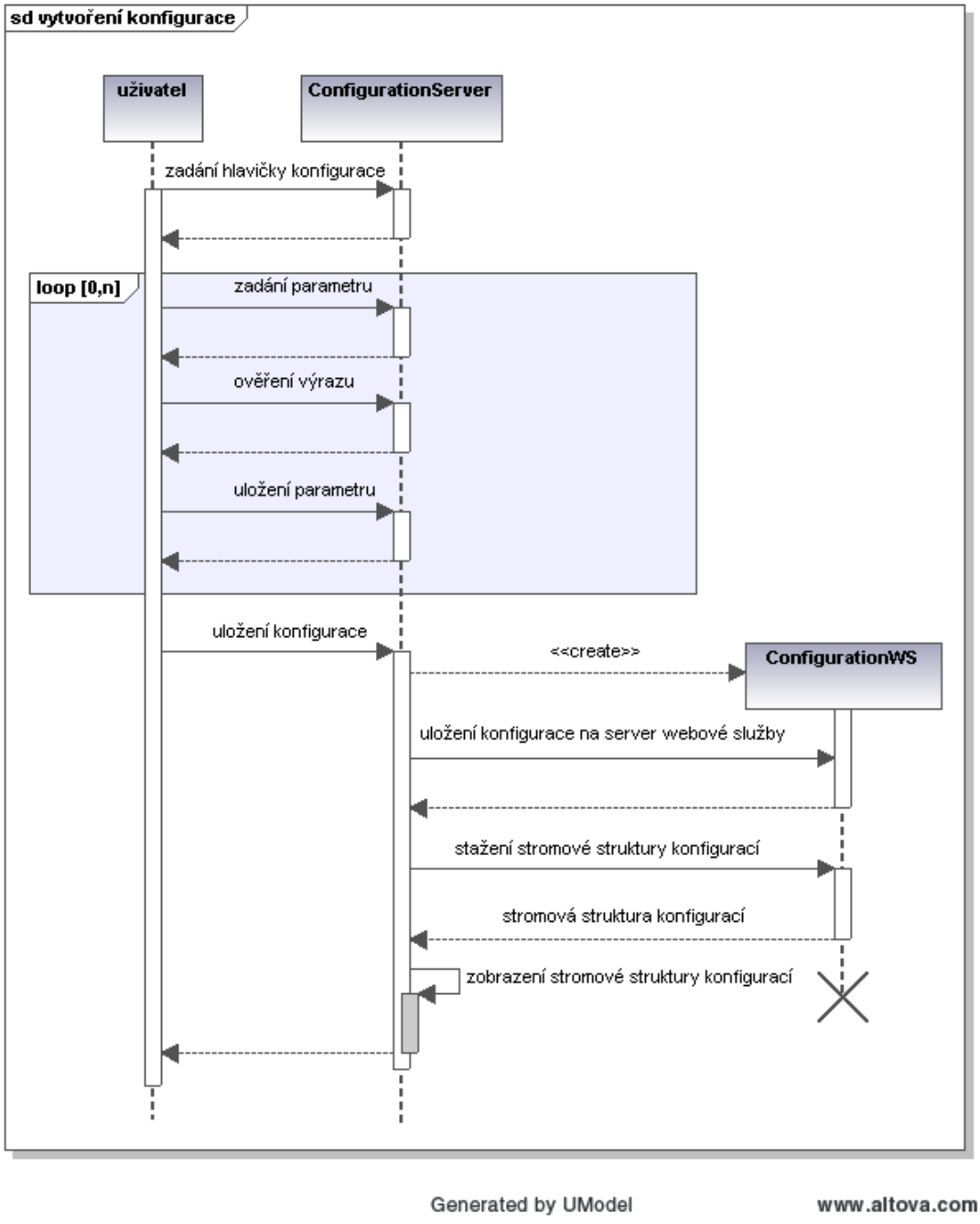
Představuje pohled na systém a jeho jednotlivé části z hlediska jeho fungování. Dynamická analýza se provede pro server, webovou službu i klienta. Detailně zpracována bude interakce serveru a webové služby. Tato interakce je zachycena na dvou sekvenčních diagramech. Jedná se o diagram popisující vytvoření konfigurace (obrázek č. 13) a o diagram znázorňující změnu konfigurace (obrázek č. 14).

6.2.1 Dynamická analýza - server

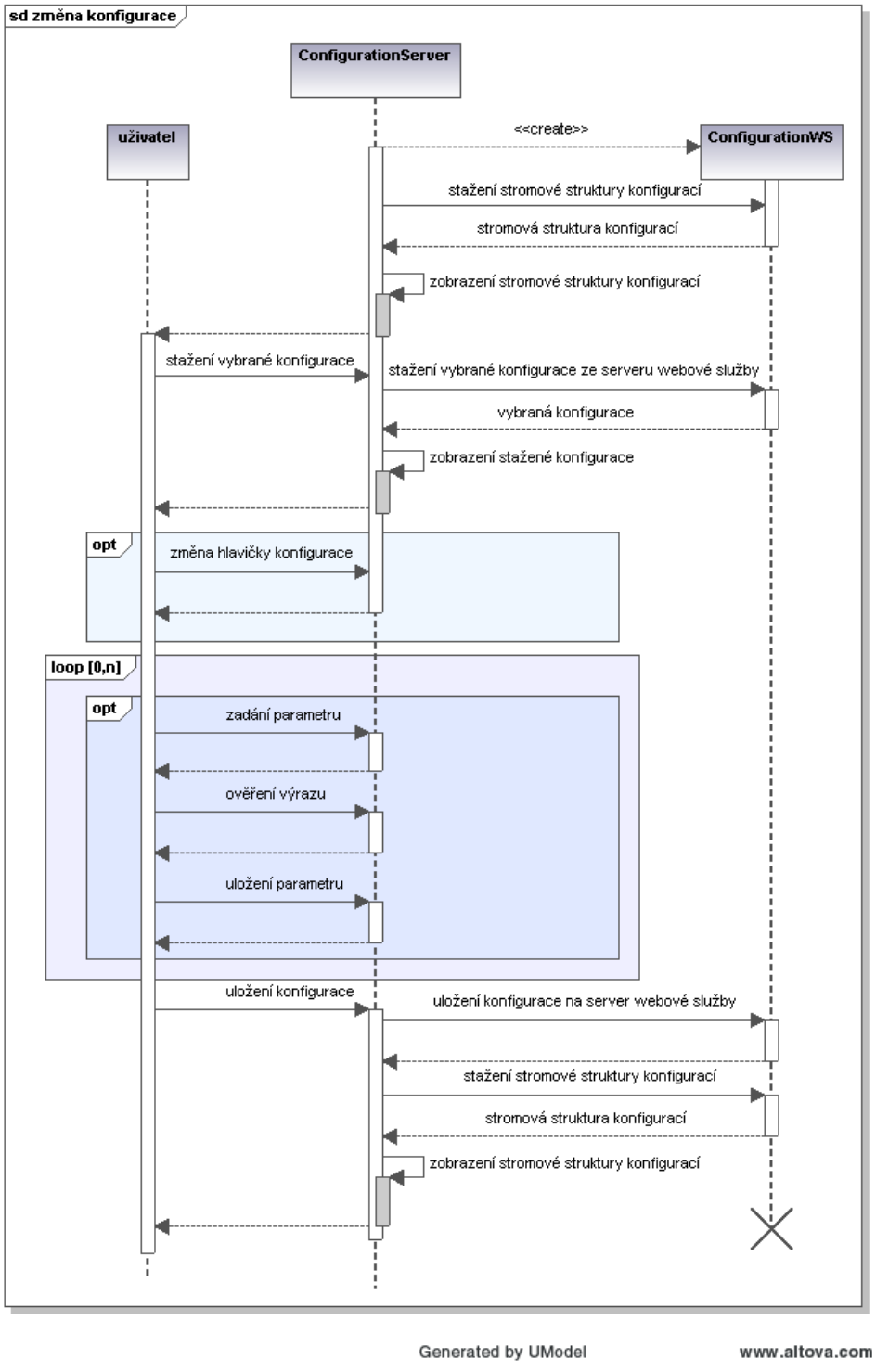
Na obou digramech jsou zachyceny obě činnosti prováděné serverem. Zde bude tedy jen doplněn stručný popis a další informace o chování serveru. Po spuštění se ze serveru webové služby stáhne strom konfigurací a ten se přehledně zobrazí v GUI komponentě, která je schopna zobrazit stromovou strukturu. Uživatel bude mít možnost si vybrat konfiguraci z kteréhokoliv uzlu, ta se stáhne ze serveru webové služby a načte do prvků grafického uživatelského rozhraní. Konfiguraci může uživatel upravit a uložit zpět. Uložit se může jako nejnovější konfigurace ve zvoleném uzlu nebo se může přehrát poslední verze. Po uložení se vždy znovu načte strom konfigurací, aby byl aktuální. Uživatel může vytvořit úplně novou konfiguraci a ji uložit stejným způsobem jako modifikovanou. Mazání konfigurací není v požadavcích a nebude ho server podporovat. Verze jednotlivých konfigurací se zobrazí ve formuláři serveru, avšak měnit nepůjdou. Verzování bude obstarávat webová služba.

6.2.2 Dynamická analýza - WS

Webová služba funguje jako úložiště konfigurací. Struktura uložených konfigurací může být načtena do objektu a zaslána serveru. Popis této struktury se nachází ve statické analýze. Místo (adresář) pro uložení konfigurace se vždy zjistí z její hlavičky. V adresáři, kam se má konfigurace uložit se nalezne soubor s verzí naposledy uložené konfigurace. Ta se buď přehraje anebo se číslo verze zvýší o jedna a uloží jako nová konfigurace v závislosti na požadavku serveru. Do konfiguračních souborů se uloží pouze parametry konfigurace. Pokud nebude existovat větev ve stromu konfigurací, pak se tato musí vytvořit. Načtení konfigurace z uzlu se provede tak, že se nalezne příslušný uzel podle hlavičky a vybere se soubor dle specifikované verze. Ze souboru se načtou konfigurační parametry a spojí se s danou hlavičkou, čímž se vytvoří kompletní konfigurace. Při požadavku klienta na konfiguraci se projde strom od kořene k cílovému listu. Během průchodu se v každém uzlu získá konfigurace. Všechny dílčí konfigurace se spojí do jediné a při shodě jmen kteréhokoliv parametru se vždy použije hodnota toho posledního. Promazávání konfigurací není třeba řešit.



Obrázek 13: Sekvenční diagram - vytvoření konfigurace



Obrázek 14: Sekvenční diagram - změna konfigurace

6.2.3 Dynamická analýza - klient

Součástí klienta bude právě jedna konfigurace. Ta bude uložena v souboru *client.config* jako záloha, ať může klient fungovat, i když nebude možné stáhnout aktuální konfiguraci. Při spuštění klienta se načte tato konfigurace. Poté se pokusí klient připojit k webové službě a stáhnout nejnovější konfiguraci. Pokud se aktualizace povede, klient normálně poběží a může se používat. Těsně před ukončením klienta se nová konfigurace uloží do zmíněného konfiguračního souboru *client.config* a tím se zálohuje.

7 Návrh a implementace

Obsahem této kapitoly je upřesnění analýzy, návrh grafického uživatelského rozhraní, popis diagramu nasazení a soupis použitých nástrojů. Na závěr kapitoly jsou zmínky o provozu a fungování systému.

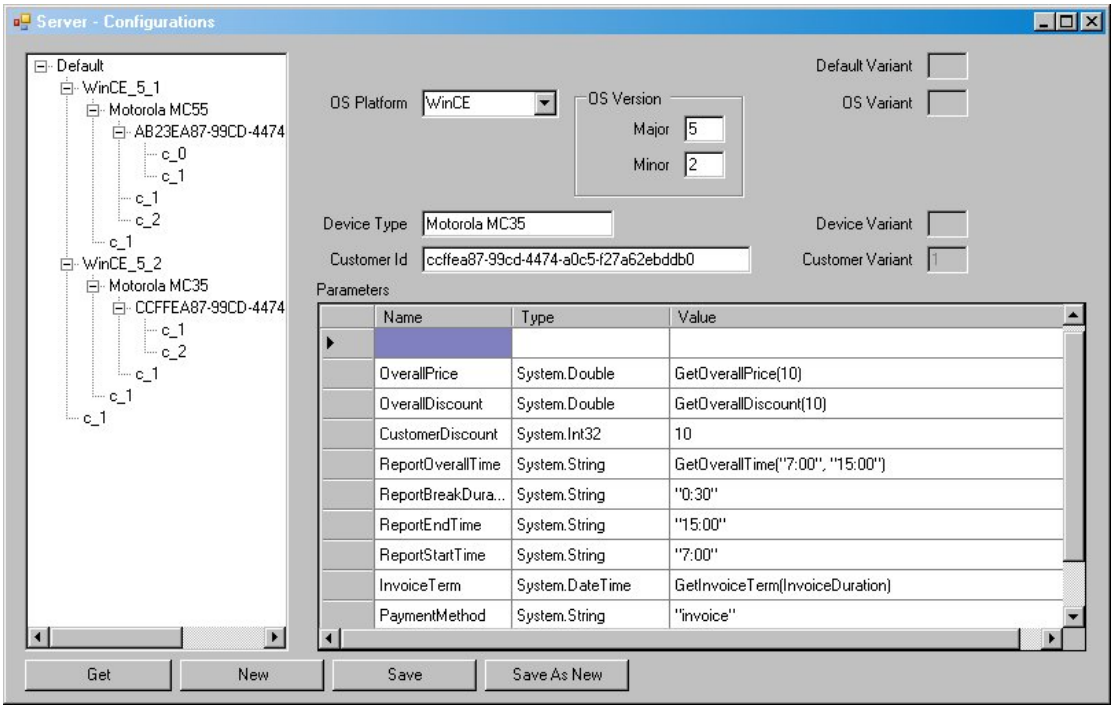
7.1 Návrh GUI

Tato sekce znázorňuje grafická uživatelská rozhraní serveru a klienta.

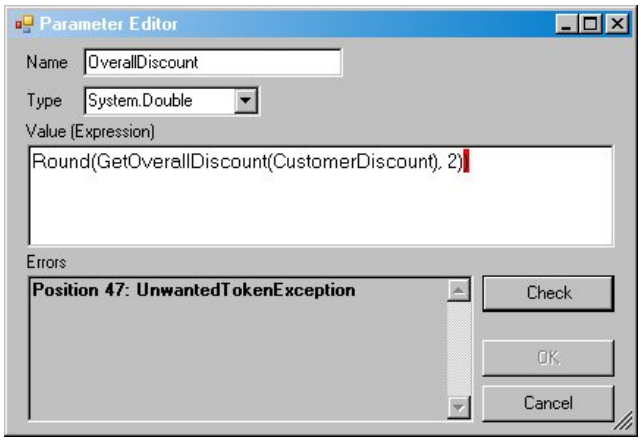
7.1.1 GUI - server

Úplně v levé části hlavní obrazovky serveru (obrázek č. 15) se nachází přehledně zobrazený strom konfigurací. V horní části jsou zobrazeny atributy hlavičky konfigurace. V této části se více vlevo vybírá platforma (*OS Platform*), verze platformy (*OS Version*) s jeho major a minor verzí. Pod volbou operačního systému se zadává typ přístroje (*Device Type*) a identifikační číslo (GUID) zákazníka (*Customer Id*). Napravo se pak nachází verze výchozí konfigurace (*Default Variant*), verze konfigurace zvoleného operačního systému (*OS Variant*), verze konfigurace zadaného typu přístroje (*Device Variant*) a verze konfigurace zadaného zákazníka (*Customer Variant*). Pod hlavičkou jsou v seznamu vypsány všechny parametry konfigurace (*Parameters*). Nejprve je jméno parametru (*Name*), pak jeho typ (*Type*) a nakonec hodnota parametru (*Value*). U spodní strany je tlačítková lišta. Tlačítka nacházející se v tlačítkové liště jsou *Get*, *New*, *Save* a *Save As New*. *Get* tlačítko se používá v případě, že se má zobrazit konfigurace z určitého uzlu stromu konfigurací. Nejprve je nutné vybrat uzel příslušné konfigurace ve stromu zobrazeném v levé části obrazovky a poté se po stisknutí tlačítka zobrazí vybraná konfigurace. *New* vymaže hlavičku i parametry, aby bylo možno vytvořit úplně novou konfiguraci. Tlačítka *Save* a *Save As New* se používají pro ukládání změněných nebo nových konfigurací. Rozdíl je mezi nimi takový, že *Save* přehraje poslední verzi konfigurace a *Save As New* vytvoří novou verzi konfigurace na serveru webové služby.

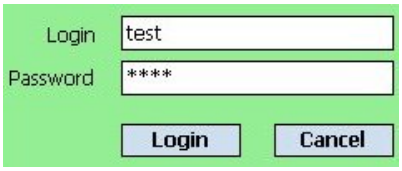
Editor parametru (obrázek č. 16) je formulář, kde se zadává jméno parametru (*Name*), vybírá se jeho typ (*Type*) z roletkového menu a vkládá se jeho hodnota (*Value*). Krom právě zmíněných prvků se na formuláři vlevo dole nachází okno s výpisem chyb (*Errors*) při ověřování výrazu zadaného jako hodnota parametru. Pokud je to možné, zvýrazní se červeně pozice, kde je ve výraze chyba. Vpravo dole se nacházejí tři tlačítka. První tlačítko *Check* je pro ověření vloženého výrazu. Jestliže ověření proběhne bez chyby, je povoleno tlačítko *OK* pro potvrzení. Posledním tlačítkem *Cancel* je možno změnu či vytvoření parametru kdykoliv zrušit.



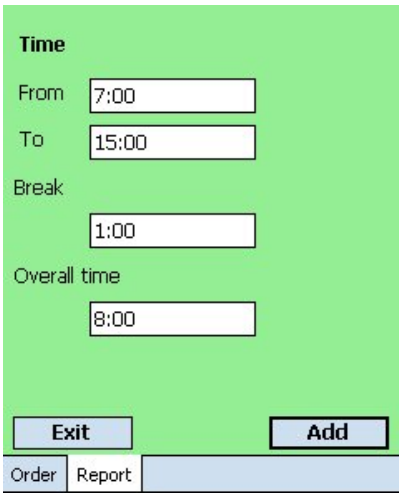
Obrázek 15: Hlavní obrazovka serveru



Obrázek 16: Obrazovka pro vytváření a úpravu parametru na serveru



Obrázek 17: Přihlašovací obrazovka klienta



Obrázek 18: Obrazovka výkazu práce na klientovi

7.1.2 GUI - klient

Na přihlašovací obrazovce klienta (obrázek č. 17) jsou dvě textová pole. Do prvního pole se zadává přihlašovací jméno (*Login*) a do druhého heslo (*Password*). Pod těmito poli je zobrazeno tlačítko *Login* pro potvrzení přihlášení a naběhnutí klienta. Vedle něj je tlačítko *Cancel* pro zrušení přihlášení a následné ukončení klientské aplikace.

Obrazovka výkazu práce na klientovi (obrázek č. 18) obsahuje odshora textová pole času od (*From*), do (*To*), délku přestávky (*Break*) a celkový čas (*Overall time*). Tato pole jsou vyplněna hodnotami příslušných parametrů z konfigurace. Tlačítko vlevo dole má za úkol ukončit aplikaci. Tlačítko vpravo dole zde nemá žádnou funkci, ale v originální aplikaci slouží k přidání výkazu práce do databáze. Pod tlačítky jsou záložky pro přepínání mezi objednávkou (*Order*) a výkazem práce (*Report*).

Obrazovka objednávky na klientovi (obrázek č. 19) obsahuje odshora textová pole množství (*Amount*), slevu na jednom kusu (*Discount per unit*), celkovou slevu (*Overall discount*), celkovou cenu (*Overall price*), způsob uhrazení fakturace (*Payment method*) a datum splatnosti faktury (*Invoice term*). Pod textovými poli je seznam výrobků, které je možno objednat. U každého výrobku je jméno (*Product*) a cena za kus (*Price*). Všechna pole krom množství jsou vyplněna hodnotami příslušných parametrů z konfigurace. Při výběru výrobku a zadání množství se vypočte celková sleva a celková cena. Tlačítko vlevo dole má za úkol ukončit aplikaci. Tlačítko vpravo dole zde nemá žádnou funkci,

Obrázek 19: Obrazovka objednávky na klientovi

ale v originální aplikaci slouží k přidání objednávky do databáze. Pod tlačítky jsou záložky pro přepínání mezi objednávkou (*Order*) a výkazem práce (*Report*).

7.2 Diagram nasazení

Schéma nasazení systému je zobrazeno na obrázku číslo 20. WS čili webová služba poběží na nějakém serveru, v tomto případě to bude lokální počítač. Tuto webovou službu budou využívat jak server, tak klient (client). Server bude spouštěn na lokálním počítači. Klient se nasadí na nějaké Windows Mobile zařízení anebo na emulátor. WS, server i klient, budou mít svou kopii společné knihovny Essence.

7.3 Použité nástroje a běhová prostředí (frameworky)

.NET Framework, .NET Compact Framework je běhové prostředí a platforma pro vývoj a sestavování aplikací s lepšími vizuálními efekty, bezproblémovou a bezpečnou komunikací a schopností modelovat širokou škálu byznys procesů.

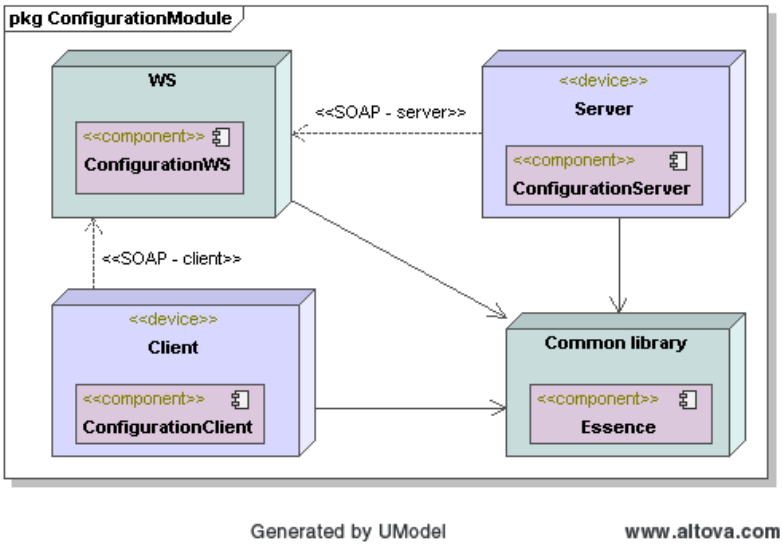
.NET Framework obsahuje:

Common Language Runtime - běhové prostředí a abstraktní vrstva nad operačním systémem.

Base Class Libraries - předem sestavený kód využitelný při řešení programovacích úloh.

Vývojové frameworky a technologie - znovupoužitelná a přizpůsobitelná řešení programovacích úloh[8].

Je to běhové prostředí, ve kterém běží všechny části systému. Server i WS běží na .NET Framework verze 3.5. Klient běží na .NET Compact Framework verze 3.5.



Obrázek 20: Diagram nasazení

C# je vysokoúrovňový objektově orientovaný programovací jazyk. Tento programovací jazyk byl vyvinut firmou Microsoft společně s platformou .NET Framework. C# je standardizovaný programovací jazyk, přičemž byl schválený standardizačními komisemi ECMA a ISO. Microsoft založil C# na známých jazycích C++ a Java (a je tedy nepřímým potomkem jazyka C, ze kterého čerpá syntaxi). Lze využít k tvorbě databázových aplikací, webových aplikací a stránek, webových služeb, formulářových aplikací ve Windows, aplikací pro mobilní zařízení (PDA, Smartphony, ...), atd.

V tomto programovacím jazyce byl naprogramován server, webová služba i klient. K programování byl použit C# verze 3.0.

Visual Studio je balík softwarových nástrojů, utilit a technologií firmy Microsoft pro snazší, rychlejší a přehlednější vývoj aplikací běžících v prostředí produktové řady Microsoft Windows. Umožňuje vyvíjet jak nativní aplikace, tak i aplikace pro .NET (Compact) Framework.

Pro snazší vývoj jakýchkoliv systémů se používají integrovaná vývojová prostředí. Rychlejší vytvoření tohoto systému bylo umožněno díky využití Visual Studia 2008.

ANTLR, ANTLRWorks je framework, který z popisu jazyka vytváří interprety, kompilátory nebo překladače v některém z velkého množství podporovaných jazyků. ANTLR umožňuje konstrukci stromu, jeho procházení, překlad, obnovení po chybě a hlášení chyb. ANTLR má sofistikované vývojové prostředí ANTLRWorks[7].

Výhody tohoto generátoru:

- Vytváření gramatiky pomocí EBNF s použitím YACC dialektu (pravidla začínají malým písmenem a tokeny naopak velkým).

- Možnost testování vytvořené gramatiky.
- Možnost ladění vytvořené gramatiky ve spojení s IDE Microsoft Visual Studio nebo Eclipse.
- Zobrazení syntaktického stromu.
- Jednoduché a přehledné IDE.

ANTLR i ANTLRWorks je možné zdarma stáhnout z url: `http://www.antlr.org/`. Na tomto webu jsou k nalezení všechny užitečné informace včetně dokumentace. Existuje také velice pěkná knížka *The Definitive ANTLR Reference : Building Domain-Specific Languages*[7], která srozumitelným způsobem popisuje základní i složitější konstrukce využívané při vytváření překladačů, stejně jako používání samotného nástroje.

V tomto nástroji byla vytvořena a otestována gramatika pro překlad výrazů. Následně byl vytvořen lexikální a syntaktický analyzátor a oba byly zabudovány do serveru. Použité verze byly ANTLR 3.0 a ANTLRWorks 1.3.1.

Json.NET je open source knihovna pro práci s JSON formátem pro výměnu dat. Tato knihovna se nachází na webu: `http://james.newtonking.com/`. Knihovna má vynikající serializaci a deserializaci. Práce s ní je jednoduchá a logická. Zpracování dokumentace k této knihovně je docela solidní.

Je to velice dobrá knihovna, ze které se použila hlavně serializace a deserializace formátu JSON. Použitá verze knihovny byla Json.NET 3.5 release 5.

Altova UModel je nástroj pro modelování a návrh softwarových systémů v jazyce UML 2. Umí generovat kód z diagramů i diagramy z kódu pro jazyky Java, C# nebo VB.NET. Umožňuje integraci s Visual Studiem či Eclipsem. UModel podporuje také modelování byznys procesů v BPMN notaci[9].

Pro modelování veškerých UML diagramů se použil tento nástroj. Verze využitá při tvorbě diplomové práce byla Altova UModel Enterprise Edition 2010.

7.4 Provoz systému

Vytvořený systém splnil požadavky a je funkční. Byl testován provozováním na lokálním počítači. Webová služba se spustila na lokálním počítači, serverem se poté upravovaly a vytvářely konfigurace. V adresáři *App_Data* webové služby je uložena stromová struktura konfigurací vytvořená během testování a krátkého provozu systému. Klient byl spouštěn pouze v emulátoru, což pro jeho plné otestování postačilo. Během krátkého provozování byl systém odladěn a byly odstraněny závažné chyby. V současnosti by měl být systém použitelný pro vytváření a ověřování konfigurací. Klient je bohužel vytvořen pouze pro omezenou množinu parametrů konfigurace a ověření konfigurací je tudíž rovněž omezené. Výkonnostní ani zátěžové testy nebyly prováděny. U systému se nepředpokládá masové používání, nebyly tedy zapotřebí.

8 Závěr

Na začátku byly specifikovány obecné požadavky, jež byly upřesněny z důvodu nejednoznačnosti řešení. Posléze vznikla analýza a návrh vycházející z požadavků. Nakonec se provedla realizace a výsledkem je vytvořený systém, který je zcela funkční a odráží zmíněné požadavky. Serverová část systému umí prohlížet, vytvářet či modifikovat konfigurace. Webová služba spravuje všechny konfigurace. A klient je v omezené míře schopen prezentovat využití.

Vlastní zkušenosti získané během vývoje se převážně týkají překladačů. Vytváření překladačů, jejich testování a zapracování do serveru vyvíjeného systému. Spolu s tím souvisí využívání ANTLR respektive ANTLRWorks nástroje. Další významná informace je seznámení se s technologií JSON. V neposlední řadě mělo výhodu použití serializace a deserializace. Bylo zde totiž třeba řešit, z mého pohledu, několik složitějších programátorských „oříšků". Získané zkušenosti jsou však velkým přínosem.

Systém je funkční, nicméně existují jistá menší či větší vylepšení, jež by se mohla realizovat. Jedno z nich se týká klienta. Pro větší možnosti využití by bylo vhodné jej více rozšířit. Mohl by mít více obrazovek i více funkcí, kde by se uplatnily větší konfigurace s větším počtem parametrů. Parametry by tak mohly vznikat komplexnější. Pak by se mohla upravit práce s konfiguracemi tak, aby se dal libovolný parametr z konfigurace použít v jiném parametru konfigurace. Tím se odstraní občas nutné redundance a sníží se riziko vzniku chyby v konfiguraci. Další z možných vylepšení je využít sofistikovanější způsob ukládání konfigurací. Místo ukládání konfigurací do adresářové struktury ukládat tyto konfigurace do SQL databáze. Poslední dvě úpravy souvisí s vytvářením konfigurací na serveru. První se týká zadávání operačního systému, typu přístroje nebo identifikace zákazníka. Nyní je nutné vše psát ručně. Lépe se jeví možnost vybrat si již z hodnot existujících ve stromové struktuře konfigurací. Druhá úprava by řešila následující problém. Jakmile se na serveru načte vybraná konfigurace, ta se modifikuje a uloží jako nová verze, pak v GUI serveru stále zůstává zobrazena původní verze. Na využitelnost má bohužel vliv pouze první zmíněná úprava a ostatní jsou jen určitá zjednodušení. Úpravy týkající se GUI nejsou zmíněny, poněvadž jsou irelevantní. Další rozšíření a úpravy se budou provádět v závislosti na požadavcích firmy KVADOS.

9 Reference

[1] KOZÁK, David. *Interval.cz* [online]. 8.10.2002 [cit. 1.3.2010]. Jak fungují webové služby. Dostupné z WWW: <http://interval.cz/clanky/jak-funguji-webove-sluzby/>.

[2] PROSISE, Jeff. *Programování v Microsoft .NET : Webové aplikace v C#, ASP.NET a .NET Framework*. Vyd. 1. [s.l.] : Computer Press, 2003. 736 s. ISBN 80-7226-879-1.

[3] BENEŠ, Miroslav. *Překladače* [online]. Ostrava : VŠB-TUO, [200?]. 123 s. Skriptum. VŠB-TUO Ostrava, Fakulta elektrotechniky a informatiky, Katedra informatiky. Dostupné z WWW: <http://www.cs.vsb.cz/behalek/vyuka/pjp/skripta/skr-mb.pdf>.

[4] GRUNE, Dick. *Modern compiler design*. Chichester : Wiley, 2000. 736 s. ISBN 0-471-97697-0.

[5] *JSON* [online]. c2009 [cit. 2.3.2010]. Úvod do JSON. Dostupné z WWW: <http://www.json.org/>.

[6] ASLESON, Ryan ; T.SCHUTTA, Nathaniel. *AJAX : Vytváříme vysoce interaktivní webové aplikace*. [s.l.] : Computer Press, 2006. 272 s. ISBN 80-251-1285-3.

[7] PARR, Terence. *The Definitive ANTLR Reference : Building Domain-Specific Languages* [online]. USA : The Pragmatic Bookshelf, 2007 [cit. 4.3.2010]. Dostupné z WWW: <http://www.pragprog.com/titles/tpantlr/the-definitive-antlr-reference>. ISBN 978-0-9787-3925-6.

[8] *Microsoft .NET* [online]. c2009 [cit. 19.3.2010]. .NET Framework Overview. Dostupné z WWW: <http://www.microsoft.com/net/overview.aspx>.

[9] *Altova* [online]. c2005-2010 [cit. 19.3.2010]. UModel - UML tool for software modeling and application development. Dostupné z WWW: <http://www.altova.com/umodel.html>.

A CD

Obsah CD:

- adresář *ANTLR*:
  - soubor *antlrworks-1.3.1.jar* - aplikace ANTLRWorks, ve které byla vytvořena gramatika.
- adresář *ConfigurationModule*:
  - adresář *Common* - knihovny ANTLR a JSON.NET nutné pro fungování systému;
  - adresář *ConfigurationClient* - zdrojové kódy klienta;
  - adresář *ConfigurationServer* - zdrojové kódy serveru a soubor obsahující gramatiku (*ConfigurationGrammar.g*);
  - adresář *ConfigurationWS* - zdrojové kódy webové služby;
  - adresář *Essence* - zdrojové kódy společné knihovny Essence.
- adresář *Dokumenty*:
  - soubor *Diplomová práce.pdf* - text diplomové práce;
  - soubor *ANTLR pro C# a Visual Studio.doc* - nastavení ANTLR a ANTLRWorks pro generování C# kódu a využití Visual Studia;
  - soubor *zprovoznění systému.doc* - popis postupu pro zprovoznění systému;
  - soubor *testovací výrazy pro gramatiku.txt* - výrazy použité pro otestování vytvořené gramatiky.